DISS. ETH NO. 28870

APPLICATION OF HETEROGENEOUS CPU-GPU HARDWARE FOR LARGE-SCALE AGENT-BASED MOBILITY SIMULATIONS

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

ALEKSANDR SAPRYKIN
Dipl., Mathematics, Saint Petersburg Electrotechnical University "LETI"

born on 27 September 1988
citizen of Russian Federation

accepted on the recommendation of

Prof. Dr. R. S. Abhari, examiner
Prof. Dr. W. Cai, co-examiner
Prof. H. Klumpner, co-examiner
Dr. N. Chokani, co-examiner

2022

ABSTRACT

The complexity of modern multi-modal transport systems increases along with the spatial scale of many high-density urban and metropolitan areas. These changes make the existing agent-based modelling tools impractical and lead to long runtimes. This work attempts to close this research gap by developing an agent-based GPU-enhanced mobility simulator (GEMSim) framework with a co-evolutionary learning loop. GEMSim incorporates a high-resolution, multi-modal and massively parallel GPU-accelerated mobility model capable of running large-scale scenarios with millions of agents in a matter of minutes. Unlike other, relatively simple and limited GPU-accelerated traffic models, GEMSim provides a modular and extensible foundation that can be used to run ever more complex future scenarios. The framework was extended to simulate public transit, on-demand coordinated fleets, and integrated Mobility-as-a-Service (MaaS) packages. The flexibility of framework's architecture allowed it to adapt and efficiently run the same traffic model on heterogeneous CPU-GPU hardware with many-core CPUs. The results show that the developed GPU-accelerated traffic model runs up to 100 times faster than one of the existing state-of-the-art mobility simulators, while the whole learning loop runs more than 22 times faster.

GEMSim was applied to quantify the uncertainties of simulation outputs when running scenarios with samples of population and downscaled infrastructure. Results show that a sample of at least 30% is recommended in scenarios with cars and public transit to accurately reproduce disaggregated outputs, and a sample of 100% is recommended for scenarios with on-demand coordinated fleets. These findings justify the need for improvements in the runtime performance of existing tools to be able to obtain accurate results.

Furthermore, GEMSim was used in multiple case studies to evaluate the potential impacts of deploying coordinated fleets in cities. First, the replacement of cars with coordinated taxi fleets in the Zurich area (Switzerland) can reduce the car ownership at the cost of the increased traffic congestion due to longer total driven distances. Second, integrated MaaS packages with public transit and coordinated fleets in the Munich area (Germany) can attract substantial numbers of public transit riders living in the areas with a relatively high connectivity; car users, however, are unlikely to switch to MaaS without push and pull measures as no significant benefits in travel time are provided. Both scenarios were created and validated using a novel agent-based demand modelling pipeline, developed as part of this thesis.

Finally, a GPU-accelerated visual analytics framework for large-scale and agent-based scenarios, Quartz, was developed and evaluated using GEMSim. Quartz is capable of performing spatio-temporal analytics on disaggregated simulation outputs with millions of agents, as well as rendering real-time visualizations of the agents' dynamics.

ZUSAMMENFASSUNG

Die Komplexität moderner multimodaler Verkehrssysteme nimmt mit der räumlichen Ausdehnung von städtischer und großstädtischer Ballungsräume zu. Für diese Fälle sind existierende agentenbasierte Modellierungen ungeeignet, da sie zu langen Berechnungszeiten führen. In dieser Arbeit versuchen wir dieses Problem zu lösen, indem wir einen agentenbasierten, GPU-gestützten Mobilitätssimulator (GEMSim) mit einer koevolutionären Lernschleife entwickeln. GEMSim bietet ein hochauflösendes, multimodales und massiv parallelisiertes, GPU-beschleunigtes Mobilitätsmodell, das in der Lage ist, komplexe Szenarien mit Millionen von Agenten in wenigen Minuten zu simulieren. Im Gegensatz zu anderen einfach gehaltenen GPU Verkehrssimulations Modellen, wurde GEMSim von Grund auf modular erweiterbar konzipiert, und kann daher für sehr komplexe Zukunftssimulationen verwendet werden. Das Framework wurde erweitert, um öffentliche Verkehrsmittel, on-demand koordinierte Flotten, sowie integrierte Mobility-as-a-Service (MaaS) Pakete zu simulieren. Dank der Flexibilität der Implementierung laufen Verkehrssimulations ebenfalls auf heterogener CPU-GPU Hardware mit vielen CPU Kernen. Unsere Ergebnisse zeigen, dass die neu entwickelten GPU-beschleunigten Verkehrssimulations bis zu 100-mal schneller sind als bestehende moderne Mobilitätssimulationen, und bis zu 22-mal schneller für die gesamte Lernschleife.

GEMSim wurde verwendet um Unsicherheiten von Simulationen zu bestimmen, die nur mit einem Bruchteil an Fahrzeugen arbeiten, und anschließend die Ergebnisse skalieren. Unsere Ergebnisse zeigen, dass bei diesem Ansatz mindestens 30% der Gesamtheit der Fahrzeuge in Szenarien mit Autos und öffentliche Verkehrsmitteln berücksichtigt werden sollte, um genaue disaggregierte Ergebnisse zu erhalten. Bei on-demand koordinierte Flotten sollte sogar die Gesamtheit der Fahrzeuge, d.h 100% berücksichtigt werden. Unsere Ergebnisse rechtfertigen daher die Notwendigkeit, die Laufzeiten von vorhandenen Tools zu verbessern, um genauere Ergebnisse zu erzielen.

Des weiteren wurde GEMSim in verschiedensten Fallstudien verwendet, um den koordinierten Einsatz von Flotten in Städten zu beurteilen: Zum einen kann der Ersatz von Autos durch koordinierte Taxiflotten in der Region Zürich (Schweiz) den Autobesitz auf Kosten der erhöhten Verkehrsbelastung aufgrund längerer Gesamtfahrstrecken reduzieren. Zweitens können integrierte MaaS-Pakete mit öffentlichem Nahverkehr und koordinierten Flotten im Raum München (Deutschland) eine beträchtliche Anzahl von Fahrgästen des öffentlichen Nahverkehrs anziehen, die in Gebieten mit einer relativ hohen Verkehrsanbindung leben; es ist jedoch unwahrscheinlich, dass Autonutzer ohne Push- und Pull-Maßnahmen auf MaaS umsteigen, da keine nennenswerten Vorteile bei der Reisezeit entstehen. Beide Szenarien wurden mit einer neuartigen agentenbasierten Nachfragemodellierungspipeline erstellt und validiert, die im Rahmen dieser Arbeit entwickelt wurde.

Abschließend wurde ein GPU-beschleunigtes visuelles Analysesystem für breit angelegte und agentenbasierte Szenarien, Quartz, entwickelt und mit GEMSim erprobt. Quartz ist in der Lage, räumlich-zeitliche Analysen von disaggregierten Simulationsergebnissen mit Millionen von Agenten durchzuführen und in Echtzeit darzustellen.

ACKNOWLEDGEMENTS

This journey started at the hill of Uetliberg and was full of challenges, inspirations, incredible ideas, unexpected discoveries, all-around emotions, success moments, and failures. The life lesson received through the years of this work is of exceptional value, and I'm grateful to everyone who shared the walk and without whom it would not have been possible.

First and foremost, I would like to express my sincere gratitude to Prof. Reza Abhari for giving me the opportunity to pursue the degree in a team of talented and amazing people. Thank you for pushing me to the limits I was not expecting from myself, for the guidance in the directions that were not obvious from the beginning, and for the freedom I had in my research. Thank you for constantly teaching me how to think outside the box, question every outcome of the work, and put practical applications at the front.

I'm deeply grateful to Dr. Ndaona Chokani for his thorough support and assistance at every stage of my work, for his insightful comments and reviews of publications and presentations, and for teaching me how to share findings with the international scientific community. Thank you for the positive mood and passion for learning that helped me look at the research results from the strong side.

I'm also grateful to Prof. Thomas Bernauer, who introduced me to interdisciplinary research and opened the door to social and political sciences – something I had never considered. Thank you for your continuous support and the opportunity to be part of ISTP, where I have learnt a lot in a great environment with fantastic people and the diversity of research disciplines.

I want to extend my thanks to Prof. Hubert Klumpner and Prof. Wentong Cai, who kindly evaluated this work as co-examiners on prompt notice.

I'm extremely grateful to Dr. Anna Gawlikowska, who believed in my potential and has always supported me throughout this endeavour. Thank you for bringing in opportunities to apply the research in practice.

I want to thank Marlene Hegner, who provided indispensable help with administrative tasks and always gave a right advice on which door to knock in every situation. Thank you for the occasional interesting discussions about many aspects of this world. I extend my thanks to Sabine Staub-Hugentobler, who also helped me with organizational matters in the last year. Many thanks to Dietmar Huber for his perfect management and support at ISTP.

Last years would not have been such a remarkable memory without my colleagues, some of whom have become good friends. Thanks to Marcello Marini for making my work possible with his intelligent agents and introducing me to the world of Nintendo. Thanks to Marco Pagani for all the epic stories, for bringing a positive atmosphere into the team, and for making the first real application of my work. Thanks to Patrik Plagowski for giving me deep insights into power systems, fruitful

discussions about optimizations in the ISTP office, and a great time in Heidelberg and South Africa. Thanks to Michael Walczak for introducing the scientific metaverse and discussing urban planning after ISTP colloquiums. Thanks to Christiaan Joubert for diving into the African culture and being brave enough to build your work on top of mine. Thanks to Marco Weber for countless talks in the LEC office and for gently pushing me towards the deadlines, even when they were not formally set. I also want to thank Patrick Eser, Annika Aurbach, Markus Brandstätter, Dominic Hänni, Jeremy Nichol and Carsten Degendorfer, and all other LEC members who were always around and ready to help.

Further acknowledgements go to ISTP members of the mobility research incubator who created and maintained a unique multidisciplinary environment. Thank you, Sergio Guidon, Michael Wicki, Fabian Willibald, Gracia Brückmann and Victor Blanco, for all our discussions, ISTP events, and for a memorable trip to South Africa. I would also like to pass appreciation to the members of other research incubators at ISTP for extending my view on many non-engineering topics in research.

Finally, I want to thank all my family members for supporting me throughout these years. My decision to do doctoral studies was a bit of a sudden, but it was understood and accepted. Thank you for your patience, love, and motivation to accomplish this challenge.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

APPLICATION OF HETEROGENEOUS CPU-GPU HARDWARE FOR LARGE-SCALE AGENT-BASED MOBILITY SIMULATIONS

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
SAPRYKIN	ALEKSANDR

With my signature I confirm that

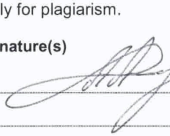
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zurich, 18.10.2022

Signature(s)



For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

NOMENCLATURE

ABBREVIATIONS

ABM	agent-based model
AI	artificial intelligence
AoS	array of structures
AMoD	automated mobility on-demand
AV	automated vehicle
BEV	battery electric vehicle
CAS	compare-and-swap
CPU	central processing unit
DCM	discrete choice model
DRAM	dynamic random-access memory
DRT	demand-responsive transport
FIFO	first in, first out
FPGA	field programmable gate array
GDP	gross domestic product
GEH	Geoffrey E. Havers (formula)
GEMSim	GPU-enhanced mobility simulator
GPU	graphics processing unit
GUI	graphical user interface
HPC	high-performance computing
ICEV	internal combustion engine vehicle
ID	identifier
IPC	instructions per cycle
MaaS	mobility-as-a-service
MAE	mean absolute error
MANE	mean absolute normalized error
ML	machine learning
MNL	multinomial logit (model)
NUMA	non-uniform memory access
OD	origin-destination

OSM	OpenStreetMap
RAM	random-access memory
RL	reinforcement learning
RMSE	root mean squared error
RMSNE	root mean squared normalized error
SAV	shared automated vehicle
SIMD	single instruction, multiple data
SIMT	single instruction, multiple threads
SM	symmetric multiprocessor
SoA	structure of arrays
SoC	state of charge
SSSP	single-source shortest path
TAZ	traffic analysis zone
TDP	thermal design power
VHT	vehicle-hours travelled
VKT	vehicle-kilometres travelled

GPU PROGRAMMING MODEL

x, y, z	coordinate components of GPU threads within block
x_b, y_b, z_b	coordinate components of GPU block within grid
D_x, D_y, D_z	dimensions of GPU block
$D_{g,x}, D_{g,y}, D_{g,z}$	dimensions of GPU grid
I_{1_3D}	ID of GPU thread in grid with single 3D block
I_{1D_3D}	ID of GPU thread in grid with array of 3D blocks
I_{3D_3D}	ID of GPU thread in 3D grid with 3D blocks

MOBILITY SIMULATION LOOP

$\mathcal{S}$	supply in mobility simulation
$\mathcal{D}_t$	demand in mobility simulation at iteration t
$\mathcal{T}$	public transit infrastructure with schedule
$\mathcal{G}$	network graph of roads and public transit routes
$\mathcal{F}$	operators of coordinated fleets
$\mathcal{S}_t^E$	output externalities at iteration t
$\mathcal{L}$	learning process of agents between iterations

$\mathcal{U}_t$	scores of daily-activity plans of agents at iteration t
P	set of daily-activity plans in agent's memory
A_k	set of activities in k-th plan of agent
L_k	set of travel legs in k-th plan of agent
V	nodes of network graph
E	edges of network graph
$\overline{E}_i$	ordered set of upstream links of i-th node
$\overline{E}^{(k)}$	ordered set of all k-th upstream links
M	maximum number of upstream links for any node
R^T	set of public transit routes
H^T	set of public transit stop facilities
V^F	set of vehicles in coordinated fleet
O^F	set of operating policies for coordinated fleet
T	number of iterations in mobility simulation
t_{sim}	current simulated time cycle [s]
t_{end}	duration of simulated iteration [s]
$M_{stats,net}$	memory for network congestion statistics [byte]
$t_{stat,net}$	duration of network statistical period [s]
$s_{cell,net}$	size of data for one statistical period per link [byte]
$M_{stats,trip}$	memory for trip statistics [byte]
$t_{stat,trip}$	duration of network statistical period [s]
$s_{cell,trip}$	size of data for one statistical period per mode [byte]
N_{modes}	number of simulated transport modes
$N_{periods}$	number of statistical periods
I_{trip}	offset for trip statistics data in output arrays

LEARNING PROCESS OF AGENTS

$U(P_k)$	score of k-th plan in agent's memory
U_{act}	score of performed activity
U_{trav}	score of travel leg
U_{dur}	score of performing activity
U_{wait}	score of waiting at activity
U_{late}	score of arriving at place late
U_{early}	score of finishing activity early

U_{short}	score of performing activity for short time
U_{best}	score of best plan in agent's memory
β_{dur}	marginal utility of performing activity
β_{wait}	marginal utility of waiting at activity
β_{late}	marginal utility of being at place late
β_{early}	marginal utility of being at place early
β_{short}	marginal utility of staying at place for short time
β_{ttime}	marginal utility of travel time
β_{tdist}	marginal utility of travel distance
t_{start}	start time of activity [s]
t_{end}	end time of activity [s]
t_{dur}	duration of activity [s]
t_{typ}	typical duration of activity [s]
t_{wait}	waiting time [s]
t_{latest}	latest possible start time of activity [s]
$t_{earliest}$	earliest possible end time of activity [s]
$t_{shortest}$	shortest possible duration of activity [s]
t_{trav}	travel time [s]
d_{trav}	travel distance [m]
p	priority of activity
$p_{i,k}$	probability to select k-th plan for i-th agent
β_U	relaxation parameter for plan selection

QUEUEING MODEL

t_{link}	minimum time vehicle stays at link [s]
t_{enter}	time vehicle entered link [s]
v_{link}	free-flow speed of link [m/s]
L_{link}	physical length of link [m]
q	flow capacity of link [vehicles/t_{period}]
t_{cycle}	duration of simulation step [s]
t_{period}	time period to specify flow capacities of links [s]
p_k	probability to select k-th upstream link
N_{lanes}	number of lanes on link
L_{veh}	average length of lane occupied by vehicle [m]

k_l	scaling coefficient of spatial buffer
k_f	scaling coefficient of capacity buffer
N_l	size of spatial buffer [vehicles]
N_f	size of capacity buffer [vehicles]
q_{off}	offset of ring buffer in global memory [byte]
q_{size}	size of ring buffer [slot]
q_{cur}	offset of first used slot in ring buffer [slot]
q_{cnt}	occupied space in ring buffer [slot]
t_{sch}	scheduled time of next update of agent [s]

VELOCITY MODEL

v_0	speed of vehicle when entering link [m/s]
v_1	speed limit on downstream link [m/s]
v_{lim}	speed limit of link [m/s]
v_{target}	target speed by end of link [m/s]
v_{exit}	achieved speed of vehicle at end of link
L	physical length of link [m]
a_{norm}	acceleration/deceleration rate of drivers [m/s^2]

VEHICLES

F_{ICEV}	fuel consumed by vehicle with internal combustion [l]
D_{km}	driven distance by vehicle [km]
f_{ICEV}	average fuel consumption of vehicle [l/km]
E_{wired}	energy consumed by non-battery electric vehicle [Wh]
T_s	driven time by vehicle [h]
p_{wired}	average power consumption of vehicle [W]
P_{wheels}	power at wheels of vehicle [W]
m	gross mass of vehicle [kg]
a	acceleration of vehicle [m/s^2]
g	gravitational acceleration constant [m/s^2]
θ	road slope [rad]
C_r, c_1, c_2	rolling resistance parameters
ρ_{air}	air mass density [kg/m^3]
A_f	frontal area of vehicle [m^2]

C_D	aerodynamic drag coefficient of vehicle
v	speed of vehicle
P_{emotor}	power at electric motor [W]
$P_{emotor,net}$	net power at electric motor [W]
P_{aux}	auxiliary power consumption [W]
P_{total}	total power consumption of electric vehicle [W]
$\eta_{driveline}$	driveline efficiency
η_{emotor}	efficiency of electric motor
$\eta_{battery}$	efficiency of battery
η_{rb}	efficiency of regenerative braking

HETEROGENEOUS COMPUTING

$I_{global,i}$	global index of i-th object in GPU memory
$I_{group,i}$	group index of i-th object in GPU memory
$I_{local,i}$	local index of i-th object in GPU memory
Q	queue of CPU-based computing tasks
T_{tot}	total number of tasks
T_{done}	number of finished tasks
CV_Q, CV_M	condition variables
M	mutex for condition variables CV_Q and CV_M
K	GPU kernel to execute on CPU
F_{GR}	green-up factor
E_C	energy consumption of CPU-based hardware [Wh]
$E_{C,G}$	energy consumption of GPU-based hardware [Wh]
P_C	power consumption of CPU-based hardware [W]
$P_{C,G}$	power consumption of GPU-based hardware [W]
t_C	runtime of CPU-based hardware [h]
$t_{C,G}$	runtime of GPU-based hardware [h]
P_{up}	power-up factor
S_{up}	speed-up factor

MULTI-MODAL EXTENSIONS

t_{fo}	delay time in flyover mode [s]
d_{fo}	flyover distance [m]

v_{fo}	flyover speed [m/s]
l_{type}	type of travel leg
f_{sync}	synchronization frequency between host and GPU [s]
f_{opt}	optimization frequency for fleet requests [s]
M_{ebuf}	GPU memory for buffering of events [byte]
E_{max}	maximum number of events agent can emit in cycle
S_{max}	maximum size of event [byte]
N_{agents}	number of agents simulated

DEMAND GENERATION

P_{in}	probability of choice i for person n
U_{in}	utility of choice i for person n
V_{in}	deterministic part of utility function (choice i, person n)
ϵ_{in}	non-deterministic part of utility function (choice i, person n)
x_{in}	vector of choice i attributes
β_i	vector of taste parameters for choice i

DOWNSCALED SCENARIOS

U	Theil's inequality coefficient
U_m	bias proportion of Theil's coefficient
U_s	variance proportion of Theil's coefficient
U_c	covariance proportion of Theil's coefficient
S_α	similarity measure with significance level α
S_α^m	traffic similarity in morning peak hour
S_α^e	traffic similarity in evening peak hour
k_l^*	optimal scaling coefficient of spatial buffers
k_l^*	optimal scaling coefficient of capacity buffers

1 INTRODUCTION

As for the future, your task is not to foresee it, but to enable it.

— Antoine de Saint-Exupery

1.1 SOCIETAL CHALLENGES

Societies are currently going through rapid and multi-dimensional transformations that are changing the way that people live and interact. At the frontier of these changes are transport systems. According to the United Nations (UN) [1], the world's population could grow from today's 7.7 billion to around 8.5 billion in 2030. Figure 1.1 shows the medium variant scenario of the population growth projected until 2100 [2] (note that LA stands for Latin America). Currently, global society is in the active growing phase of population development, and this process of the population growth is expected to continue until about 2050. While after 2050 most of the geographic regions are expected to have declining populations, Africa can still drive the increase of global population.

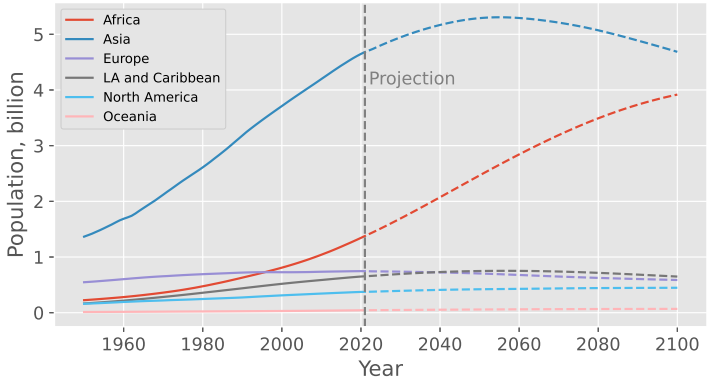


FIGURE 1.1: Historical and projected data for global population growth in 1950–2100, medium variant scenario from the UN.

At the same time, more and more people will move to urban areas; indeed, since 1950, the share of urban population has increased from 30% to 55%, and is expected to increase further to 68% by 2050 [3]. This population growth, along with urbanization and city sprawl, imposes new challenges for transport engineers and city planners, who seek to adapt existing infrastructure for increased demand in

order to provide efficient accessibility. For example, according to Urban Mobility Report 2021 [4], the United States of America (USA) had in 2019 a congestion cost of US$190 billion, which includes about 8.7 billion travel delay hours (54 hours annually per commuter) and 13.25 billion litres of wasted excess fuel. The report also demonstrates that these negative economic impacts of the over-used road transport infrastructure increase non-linearly with regards to the urbanisation and the population growth processes: in 1982, the congestion cost was only about US$15 billion (inflation-adjusted to 2020), more than 22 times lower. In the European Union (EU), according to the report of the European Court of Auditors [5], inefficiencies in urban mobility, and the negative impacts of traffic congestion on the roads, cost the EU about €110 billion, which is more than 1% of the EU's gross domestic product (GDP). Moreover, as about 70% of Europe's population already lives in urban areas (and that number is expected to reach 80% by 2050 according to UN forecasts [3]) the impacts of inefficiencies in urban mobility will become even greater without adopting relevant policies. China, currently the world's most populated country with about 1.5 billion inhabitants, experiences the same negative impacts of growing population in urban areas onto transport systems. According to the Ministry of Transport of China, the annual economic loss from traffic congestion is about 250 billion yuan (about US$36.25 billion by the current exchange rate) [6]. As mobility is always one of the key drivers of economic growth, it is of the utmost importance for any society to align the development of transport systems with the needs of the population and economy.

Aside from natural population growth and urbanization, a broad range of other societal challenges further affects the transformation of the existing mobility infrastructure. First, there is a trend in many societies for improved sustainability, which entails the more responsible use of natural resources, like fossil fuels and minerals, and reducing the negative externalities, like noise and air pollution, affecting the environment. Since 1993, KPMG, a multinational professional services network, has been surveying sustainability reporting among global companies, and the most recently published survey [7] shows that the number of reporting companies is growing. Figure 1.2 shows the rate of sustainability reporting companies since the first survey was done. One sample of the companies, G250, refers to the world's 250 largest companies by revenue; and another sample, N100, refers to a worldwide set of 5 200 companies, using the top 100 companies in each of the 52 countries and jurisdictions where the research was conducted. From the survey, one can note the upward trend since the previous report made in 2017. While the largest companies are typically leaders in sustainability reporting (95% in 2020), smaller companies are now adopting the same practices worldwide (80% in 2020). In the N100 sample, North America has the highest regional reporting rate of 90%, while Europe stays at 77%, with Eastern Europe a key contributor to the growth of the reporting rate. Interestingly, Japan and Mexico reached a 100% reporting rate for their top companies in the sample. Another interesting aspect of the survey is the strong growth of the number of companies reporting UN Sustainable Development Goals (SDGs) [8], which is a set of 17 interlinked goals targeted towards the peace

and prosperity of the planet. Since 2017, the number of SDG-reporting companies rose from 39% to 69% in the N100 sample, and from 43% to 72% in the G250 sample.

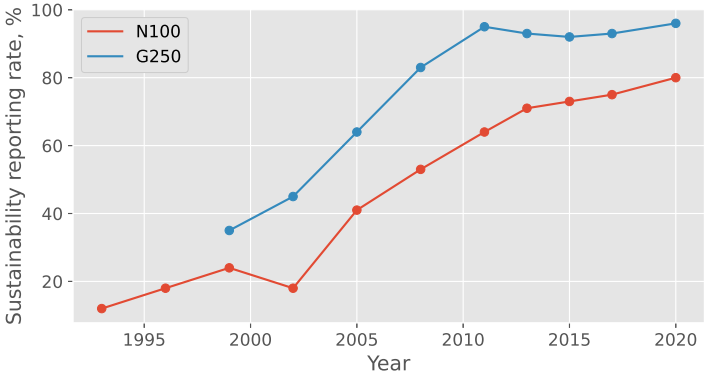
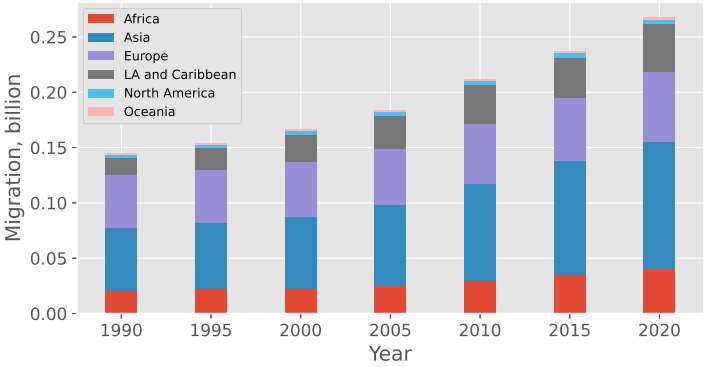


FIGURE 1.2: Global sustainability reporting rates of companies from two samples: largest by revenue (G250) and a broad-based set (N100) of large and mid-cap companies.

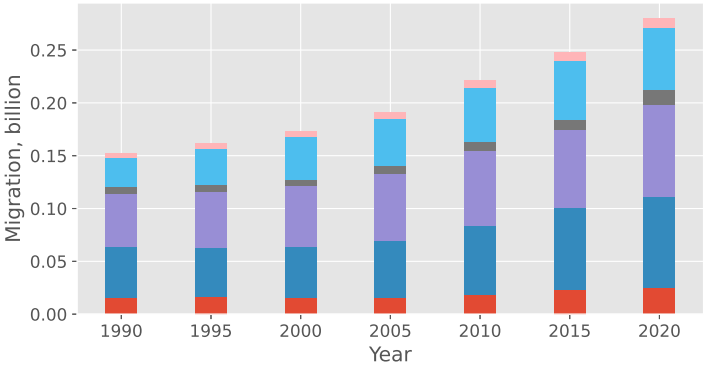
The transport sector attracts more attention from societies regarding sustainability and negative impacts on human health. It is estimated that global transport emissions in 2015 contributed to about 385 000 premature deaths [9], mostly in the top global vehicle markets, including the USA, EU, China and India. There is also evidence that noise pollution negatively impacts human health [10, 11].

Migration is another challenge experienced today by many societies across the globe. According to the International Organisation for Migration [12], there have been about 272 million international migrants (or 3.5% of the world's population) in 2019, which is almost a two-fold increase from 150 million in 2000. Figure 1.3 shows the global migration trends for the last 30 years [13] by geographic region of origin and destination. The main sources of migrants are Africa, Asia, Europe and Latin America and the Caribbean. In Europe, however, many people move internally, hence the outflow is relatively small. Interestingly, the North America region has almost negligible outflows, while at the same time about 50 million migrants are there. In overall, more developed regions like Europe and North America attract more migrants, with Asia also having an increased number of internal migrants. Migration can impact urban development strategies through the densification and sprawling processes, which can cause marginalization and the reduction in sustainable development [14]. In particular, dense urban areas may require an increase in the capacity of the local road network, while sprawling areas may negatively impact the cost of infrastructure [15] and increase the amount of negative externalities generated [16].

Third, technological improvements in communications and the improved accessibility of the population to global networks put mobility service providers closer to customers. This proximity allows creating new business opportunities like ride-



(a) By origin.



(b) By destination.

FIGURE 1.3: Number of migrants by a geographic region of origin and destination.

hailing and micro-mobility with shared e-scooters and e-bikes. To give an example, Figure 1.4 shows the number of monthly active platform consumers (MAPCs) in Uber [17–19], one of the largest ride-hailing companies in the world. An MAPC is a unique client who took a ride or used a food delivery service in a given month. One can see rapid adoption of new ride-hailing technologies in four years preceding the COVID-19 pandemic, when the number of MAPCs grew from 19 million in 2016 to 111 million in 2019. After a strong drop down to 55 million MAPCs in 2020 due to the pandemic, the number of consumers started growing again after many countries lifted movement restrictions imposed during the pandemic, and in 2021 the number of MAPCs exceeded pre-pandemic levels. This rapid adoption of ride-hailing technologies was possible thanks to the wide accessibility of technologies like the Internet and mobile phones. Technological improvements also lead to the development of smart infrastructure, where its elements (including vehicles) are connected into a network in order to optimize overall operating efficiency. For exam-

ple, adaptive traffic lights with applied reinforcement learning (RL) can outperform non-adaptable traffic light controllers [20].

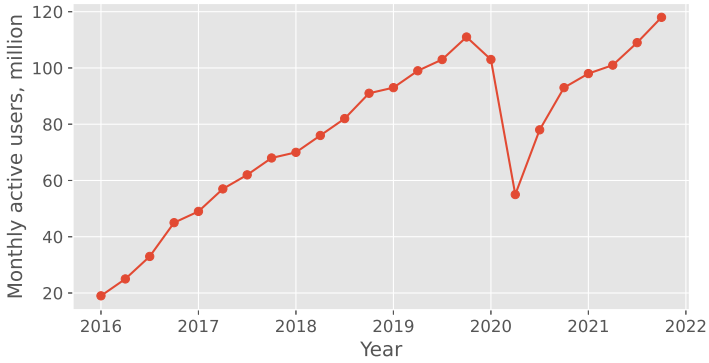


FIGURE 1.4: Monthly active platform consumers in Uber, with a recovery trend after the COVID-19 pandemic.

1.2 MOBILITY TRANSFORMATIONS

It is crucial to understand and foresee the implications of ongoing and upcoming changes in mobility in order to perform the transition in the most efficient and beneficial way for a society. However, the aforementioned social challenges lead to the increase of the complexity of modern mobility systems in order to efficiently serve the demand from the increasing population in urban areas and to ensure compliance with policies that support sustainable development and the energy transition [21, 22]. Emerging technologies such as battery electric vehicles (BEVs) and automated vehicles (AVs) are among the main drivers of the ongoing transformations in the transport sector [23, 24]. Figure 1.5 shows global energy-related CO_2 emissions [25], which reached 34.8 Gt in 2020; it is estimated that the global transport sector is responsible for about 8.7 Gt of CO_2 emissions, or 25% of the total volume [26]. That is why the mobility transition from fossil fuels to alternative fuels can make a major positive contribution in the reduction of CO_2 emissions.

BEVs and AVs require new infrastructure (such as chargers for BEVs and intelligent traffic management systems for AVs) to be built and maintained [27]. The increased penetration of BEVs and AVs is also paralleled by changes in the behaviour of people, due to facts like BEVs requiring additional time for charging, BEVs having limited range compared to internal combustion engine vehicles (ICEVs) [28], or cheaper AVs result in a different perception of the value of time [29]. Figure 1.6 shows new registrations of BEVs and plug-in hybrid electric vehicles (PHEVs) in the USA, Europe (the EU with Norway, Iceland, the United Kingdom (UK) and Switzerland) and China in recent years [30]. While the magnitude of

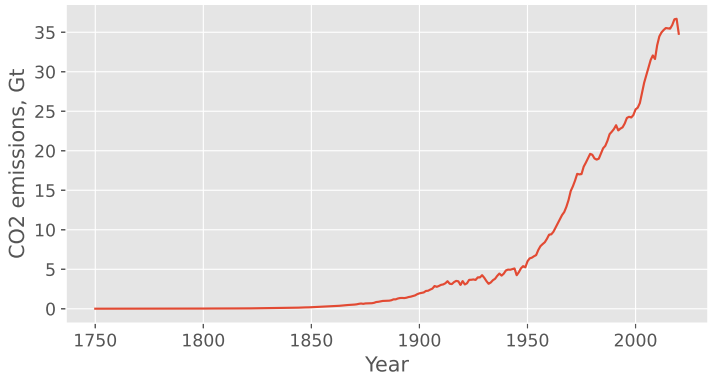


FIGURE 1.5: Global energy-related CO_2 emissions in 1750–2019.

electric car registration varies across the regions, there is a consistent trend towards the increase of BEV and PHEV sales. The share of electric cars in total sales reached 16% and 17% in China and Europe, respectively, while in the USA the share stays at a relatively low level of about 5%.

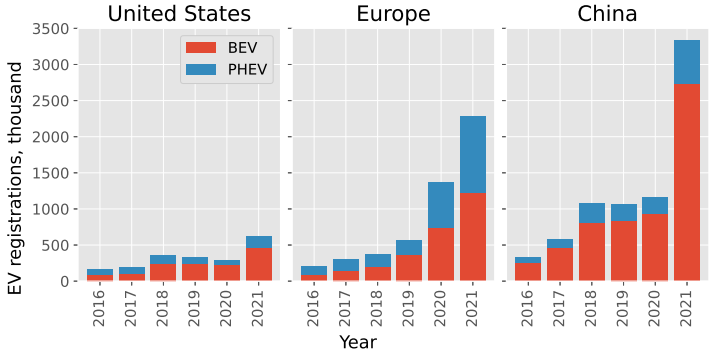


FIGURE 1.6: Electric car registrations in the USA, Europe and China.

As further reductions in mobility-related CO_2 emissions are targeted by many countries [31], it is anticipated that the transition to EVs will intensify in the near future. Furthermore, traditional and free-floating car-sharing services are expected to reduce the cost of car ownership [32] and to have a positive impact on sustainable travel [33] by changing patterns of behaviour. The increasing penetration of BEVs also leads to the coupling of mobility and grid infrastructures as charging demand at certain locations requires a grid operator to provide sufficient power [34]. Similarly, the vehicle-to-grid technology allows BEVs to participate in electricity trading using on-board batteries [35]. These integration processes will drive the future

development of mobility, and shall, therefore, be given special attention by city planners and policy-makers.

Simultaneously, and more importantly, there is a trend towards a closer integration of infrastructure and services of different transport modes. The Mobility-as-a-Service (MaaS) concept [36] integrates multiple forms of mobility services into a single service, trying to optimize available modes in order to provide, on demand, the best-value option for customers. This way, sustainable travel can be developed further by providing a viable, convenient and cheaper alternative to cars. The UbiGo project in Gothenburg (Sweden) was a pioneering MaaS concept in which a six-month trial was performed using paid subscription model [37]. The service included customized travels solutions with public transit, taxi, car- and bike-sharing. The overall outcome of the experiment was positive, and 97% of the participants wanted to continue with UbiGo. The participants also became more positive towards non-car modes of transport. Another example of MaaS development and implementation is the Whim start-up from Finland [38], which provides services in the Helsinki metropolitan area. Whim provides packages with public transit, taxi, car rentals, bikes and shared micro-mobility like e-scooters. The authors note, however, that, aside from the technology, MaaS solutions require support and coordination from various stakeholders in order to make such transport services commercially successful.

1.3 TRANSPORT MODELLING

Given the complexity of emerging mobility systems, it is almost impractical to analyse their possible transformations using empirical experiments. Moreover, the stochastic nature of people's behaviour and the scale of metropolitan urban areas do not allow the application of analytical solutions without serious limitations. Mobility simulations offer one approach to improve planning decisions and provide more cost-effective assessments of potential solutions, including policies.

The first traffic simulation models date back to the 1950s [39, 40], when theoretical developments of traffic simulation were established by then-emerging transport engineering. In that decade, one of the currently most widely used transport forecasting models, the four-step model [41], was developed to run on mainframe computers. The model comprises the following steps:

- **Trip generation** determines the number of outgoing and incoming trips for each traffic analysis zone (TAZ).
- **Trip distribution** produces an origin-destination (OD) matrix by matching origins with destinations.
- **Mode choice** determines the share of trips for each OD pair that uses a specific mode.
- **Route assignment** determines a per-mode route for each OD pair.

While the four-step model is easy to understand and effective for static planning (without accounting for temporal effects like congestion spillover) and for simple scenarios, as discussed elsewhere [42, 43], the four-step model is less realistic and less precise for dynamic planning, as a person's decision-making process is not well represented; this is especially so when the extent of the decision space is large. Four-step models are also impractical for modelling emerging transport modes that require tight coupling of city infrastructures, as these modes heavily rely on individual behaviour. Rather than using an aggregated approach for behaviour, agent-based models (ABMs) are used to simulate the behaviour and interaction of people in complex urban environments [44–48]. In ABM, agents are considered as individuals with their own behavioural logic and rules of interaction; thus, ABM simulations are more realistic, and provide a framework to understand causal effects in the complex system as a whole. ABMs are not limited to humans but cover a broad range of disciplines, from particles in physics to living species in biology [49]. Typically, in a mobility ABM, each agent represents a human being or a vehicle within a simulated environment comprised of roads, public transit systems and other mobility services, as well as the policies and regulations that influence agents' decision-making. During simulation, agents can react to events and adapt their behaviour. For example, if a road is blocked because of an accident, then agents can re-route to reach their intended destinations.

On the one hand, ABM yields mobility simulations with a high level of detail and complexity in which behaviour and decision-making are driven by the same factors as in the real world (for example, congestion, road accidents, public transport delays, social networks, and locations of places of activity). Thus, very realistic scenarios can be formulated and the impacts of potential changes in policies, infrastructure and transport services can be accurately assessed at the spatial and temporal resolution of each simulated agent. On the other hand, large-scale and detailed agent-based models – which integrate complex behaviour and rules of interaction – impose a substantial computational burden on the simulation process. ABM applied to large metropolitan areas may encounter bottlenecks in computing performance with correspondingly long simulation runtimes; these challenges reduce the attractiveness of such models for transport planners and policy-makers. Moreover, the cost of the required hardware is also a burden when high performance is needed.

A broad range of work, from multi-threaded processing to distributed computing on clusters, has been conducted to improve the scalability of traffic simulations. Multi-threading is typically employed [50, 51] as a first step to accelerate simulations using multi-core central processing units (CPUs). However, due to limitations in the available hardware resources (such as number of CPUs, or amount of shared memory) of a single computing node, the performance improvements are limited. Distributed computing offers more opportunities to improve the scalability of traffic simulations [52–56] on large multi-node clusters, but this requires that the simulation models be adapted. In order to run large-scale simulations on multiple cluster nodes, the simulation domain must be decomposed [57–59] and evenly distributed across the nodes using load-balancing techniques [60]. Simultaneously,

the execution of the simulation must be synchronized [61] across the nodes, and network communications [53] must be reduced. Surprisingly, distributed multi-modal traffic simulations are not widely reported, and have only recently received some attention [62, 63]. Some attempts have been made to improve computing performance using more efficient algorithms [64, 65] or data structures [66]. Despite the improvements made with such approaches in recent years, it seems that the complexity of simulated transport systems is increasing at a rate faster than that at which performance improvements come into place. Another limitation of many works is a pure focus on traffic simulations rather than on the mobility system as a whole, which reduces the applicability of such limited models for urban transformation scenarios; today, the simulation of car traffic is only a subset of the capabilities required of mobility modelling tools.

Therefore, another approach adopted by researchers is, first, to downscale the demand and supply of mobility in the simulated scenario: a fraction of agents is randomly sampled from the full population, while the capacity of the road network (that is, the rate at which vehicles pass through a transport link) and other infrastructure are also downscaled. The simulation outcomes are then scaled up, and these results are considered to be a good approximation of the full population and the actual network. For example, if a 10% sample of the whole population is used, then the capacity of the roads is also downscaled to 10% (approximately; some variation is possible due to calibration). The simulation outcomes are then scaled up by a factor of 10. Basically, each agent in the downscaled scenario represents an aggregate group, not each individual, of the real world. However, the impacts of such downscaled inputs on the scenario outputs are not well studied.

1.4 COURSE OF HETEROGENEITY

According to the well-known Moore's law [67, 68], the number of transistors in a dense integrated circuit doubles every two years. In general, since the initial prediction made in 1965 and refined in 1975, the increased number of transistors in a CPU means increased performance, and traffic simulations should not be an exception. Figure 1.7 shows historical data for CPU developments in the last 50 years [69]. One can clearly see that Moore's law is still relevant as the number of transistors in a CPU continues to grow. However, the relative gap between the increasing transistor count and the performance improvement of a single CPU core is becoming larger. This can be simply explained by the fact that the frequency of a CPU core stopped increasing from around the year of 2010 due to physical limits and circuit stability, and, at the same time, the number of logical cores started increasing. Therefore, in order to get performance improvement in simulation models when using modern multi-core CPUs, one has to parallelize the code.

However, the parallelization of the code can be a non-trivial task, depending on the software architecture and the nature of the problem solved with the code. First, not every code can be parallelized, limiting the applicability of multi-core

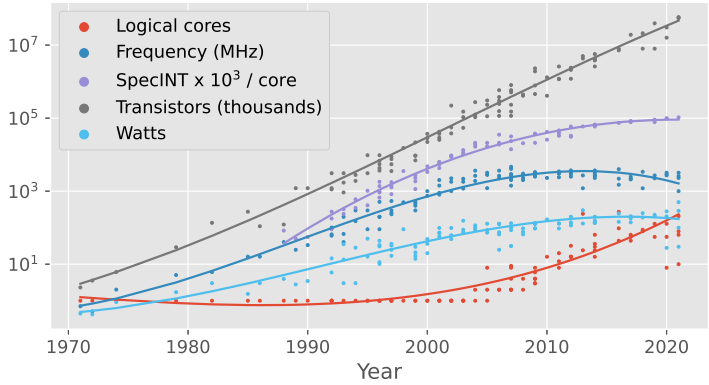


FIGURE 1.7: Evolution of CPU specifications in the last 50 years, notable increase of the number of logical cores and the decline of operating frequency.

CPUs for existing models. For example, when some algorithms and tasks are executed sequentially with a single CPU core, a multi-core CPU will not bring runtime benefits. Second, even if some parts of an application can benefit from the execution on a multi-core CPU, the overall improvement in runtime is limited by the fraction of time the parallelized code is used. This is known as Amdahl's law [70]. Moreover, as the overall computing performance of CPUs did not grow substantially in the last years, it leads to the conclusion that capabilities of CPUs to improve runtime performance of existing computation-intensive models within a single node are already almost exhausted. As mentioned above, distributed computing together with new algorithms do not always provide the required pace of improvements. This is one of the reasons why researchers are looking into the computing capabilities of other types of hardware like graphics processing units (GPUs) and field programmable gate arrays (FPGAs).

Figure 1.8 shows the trends in computing performance (floating point operations per second, FLOPS) of some CPUs and GPUs on the market [71], including some announced as of 2022. While the performance of CPUs has grown slowly, GPUs show an up to five times greater increase in raw computing performance. This performance leap has led to a wider adoption of GPUs in many fields of academia and industry, making these devices more accessible. Figure 1.9 shows the number of GPU-accelerated computer systems in the TOP500 [72] list, and today about 28% of systems have GPUs installed. Moreover, initial exascale computer systems are expected to be GPU-accelerated [73], as purely CPU-based systems cannot deliver the same raw performance within the same power and price envelope. This is one of the reasons for the increased hardware heterogeneity in recent years.

Another reason for hardware heterogeneity is the increased use of cloud technologies with the diversity of the workloads customers are running in clouds. Some estimations show that the global cloud computing market size could double from

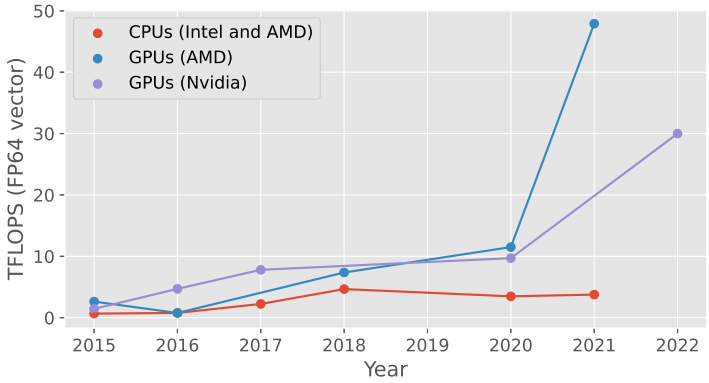


FIGURE 1.8: Comparison of FP64 (floating point, 64-bit) performance for selected top CPUs and GPUs. In recent years GPUs provide exponential growth in performance while CPUs are stagnating.

US$445 billion in 2021 to US$947 billion by 2026 [74]. This is happening not only due to the increased demand for digital services during the COVID-19 pandemic, but also because of the increased use of artificial intelligence (AI) and machine learning (ML) technologies by many companies seeking to optimize operating costs and the scalability of their infrastructure. It should also be noted that today, ML technologies are infused into traditional high-performance computing (HPC) simulations when data is available, therefore making a need for GPUs and other custom accelerators. For example, mobility simulations may now include RL, and this type of workload can benefit substantially from executing on GPUs. Hence, the need for hardware accelerators comes not only from the existing and established models, but also through the intensified penetration of other technologies. The increased availability of heterogeneous hardware, in industry and in HPC sectors, opens new opportunities for the simulation models that can utilize computing power more efficiently. Heterogeneous hardware is likely going to dominate future high-performance systems as a key driver in their performance growth; therefore, in order to bring mobility simulations at the next level, the models shall be re-developed for the changing environment.

1.5 LITERATURE REVIEW

Currently, the transport sector is in transition from a set of relatively disjointed systems to a highly integrated and optimized system as a whole. This not only includes various modes of transport, but also the whole complexity of human behaviour. Transport modelling is one of the key methods for understanding the impacts of ongoing transformations in society on mobility and vice versa, and can also help city planners and policy-makers to take decisions for robust and efficient

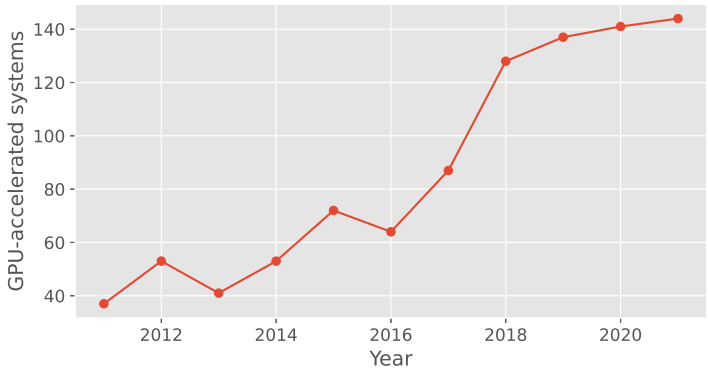


FIGURE 1.9: Number of GPU-accelerated computer systems in the TOP500 list displaying an upward trend, although flattening in recent years.

solutions. Along with the transformations in mobility, simulation models have to factor in the relevant processes to match the needs of decision-makers and engineers. Taking into account the increasing scale of simulations and the increasing complexity of the models, improvements in the methodology of agent-based mobility modelling are required in order to make simulated scenarios practical (i.e., of sufficient scale and with acceptable runtimes). However, the traditional CPU-based hardware used for traffic simulations has stagnated in computing power per core and provides limited capacity for the necessary improvements in models' runtime performance. Limitations imposed by distributed computing are especially tangible when it comes to the modelling of emerging transport modes like coordinated fleets or shared micro-mobility services. This section provides a general review of the literature, while each chapter provides a more detailed literature review where it is relevant.

One of the still under-explored directions of possible substantial improvements in the runtime performance of ABMs in mobility is the use of hardware accelerators. Previously, some attempts have been made to utilize GPUs and FPGAs for traffic simulations, but the models developed were either not scalable or limited in functionality. Strippgen and Nagel [75] performed one of the first attempts to run agent-based traffic simulation on a GPU using a queue-based mesoscopic model, and speed-up factors of 5.5 to 67 were achieved. The developers, however, faced issues while integrating this GPU-accelerated model into MATSim [76]. The queue-based model was relatively simple, without public transit support and gridlock resolution, but proposed data structures for GPUs were important contribution into the field. Perumalla et al. [77] implemented a field-based GPU-accelerated mobility model and applied it for a large-scale scenario of the state of Texas (USA), with the network of 2 million nodes and 5 million links, and initially 10 vehicles per node. Results showed that is takes 13 minutes to run this scenario, but serious limitations in modelled behaviour of the agents were applied making the model

more suitable for evacuation scenarios. Shen et al. [78] developed a microscopic traffic simulator with GPU acceleration and used it to optimize traffic signal timing. The simulator, however, was not suitable for large-scale scenarios while focusing on the specific optimization problem. Wang and Shen [79] presented a microscopic traffic simulation model running on a GPU with a whole loop including the learning process between iterations. While the model achieved a speed-up factor of about 105 compared to a CPU-based version, it was tested only on small lattice networks. Moreover, other limitations were applied, like a single trip per agent and the lack of multi-modality. Hirabayashi et al. [80] proposed multiple design schemes of traffic simulation models on a GPU, and the results showed that a better runtime performance can be achieved if more work is pushed to a GPU with less CPU-GPU synchronization. Only one-dimensional version of the optimal velocity model [81] was implemented, but it was able to run about 1 000 times faster on a GPU compared to a single-threaded CPU version. Sano and Fukuta [82, 83] developed a framework for large-scale agent-based simulations with the focus on traffic. The works, however, were mostly focused on the agent-based daily activity planning, including routing algorithms. This framework was later extended by Sano et al. [84] to improve semi-automated performance tuning of the GPU-accelerated code. Xu et al. [85, 86] developed a GPU-based microsimulation framework called Entry Time-based Supply Framework (ETSF). The framework proposed a new method to reduce the time spent on processing vehicles during their movement. The framework showed a factor of 11.2 speed-up compared to a single-core CPU model, however only a small-scale artificial grid network (10 201 nodes and 20 200 links) was used. On a network with the realistic topology of Singapore (3 179 nodes and 9 419 lanes) the framework showed a factor of 2.37 speed-up [87]. Vu and Tan [88] developed a GPU-based mesoscopic traffic simulation framework. Depending on simulated static demand (100 000 to 300 000 vehicles) in the Singapore area, a factor of 5.8 to 6.5 speed-up compared to a single-core CPU-based version was achieved. Later, the framework was extended [89] with demand simulation on a GPU with yet serious limitations like predefined sets of routes for OD pairs. Heywood et al. [90] used the FLAME GPU framework [91] to run mobility simulations of very limited scale and capabilities. For example, agents moved randomly on an artificial grid network without any learning process implemented. Later, the model was improved for transport simulations [92], and a factor of 43.8 speed-up compared to the multi-threaded commercial microscopic simulator Aimsun [93] was achieved, keeping most of the original limitations. Recently, Yedavalli et al. [94] developed a GPU-accelerated microscopic traffic simulator called MANTA which was able to run a scenario for San Francisco with 3.2 million trips in about 4.6 minutes. The simulator had some serious limitations, for example, the lack of learning process and non-car transport modes, as well as a static demand model with a single trip per agent. A more comprehensive survey on the application of hardware accelerators in agent-based models is available elsewhere [95].

The use of heterogeneous hardware can provide a natural way to reduce the run time of mobility simulations, and this way can bring even more benefits as

the use of hardware accelerators is growing in many mobility-related fields, hence, the technology can be utilized later in even more sophisticated simulations. Up to now, only a few attempts have been made to implement traffic models capable of running on heterogeneous hardware. Xiao et al. [96] offloaded parts of a CPU-based microscopic simulator to a GPU using the OpenCL framework. A fully offloaded GPU-based version was found to deliver the highest speed-up factor of 28.7 compared to a sequential CPU-based version. The authors noted that a fully OpenCL-based implementation was challenging to develop, had limited maintainability and extensibility of the model, and that many commonly used data structures had to be re-implemented for hardware accelerators. Later, Xiao et al. [97] applied an OpenCL-based microscopic agent-based simulation model to small-scale (16 384 agents on a single four-lane road) traffic simulations executed on CPUs, GPUs and FPGAs, where the FPGA-based implementation delivered the shortest runtime among other types of hardware. Rajf and Potuzak [98] implemented and compared the runtimes of two microscopic traffic models on CPUs and GPUs. While the models were implemented in different programming languages, a speed-up of up to factor of 12.4 was shown for the GPU-based model compared to the CPU-based multi-threaded model. In other studies on hardware-accelerated traffic models [75, 79, 88, 99] only single-threaded limited implementations for CPUs were evaluated. The literature shows that previous works on traffic models which can run on heterogeneous hardware mostly rely on the OpenCL framework (with its own limitations), none of the works demonstrate multi-modal scenarios, and CPU-based models were limited in scalability.

Hence, it remained an open question if specialized hardware accelerators can not only bring some performance improvements in prototyped traffic models, but rather be used for practical applications for large-scale and complex multi-modal scenarios. While the simulation of a public transit system is an essential part of many existing mobility simulators [76, 93, 100–102], the modelling of this transport mode on high-performance systems received a very little attention in the literature. Only a few works demonstrated multi-modal distributed mobility simulations [62, 63], as well as a CPU-based parallelized simulation of public transit [103]. In the field of large-scale modelling of coordinated fleets, recent works also emphasize the poor performance of existing simulation tools. Bischoff and Maciejewski [104] simulated the replacement of private cars in Berlin (Germany) with a coordinated taxi fleet varying its size from 50 000 to 250 000 vehicles. The authors reported that it took about 3 hours to run a single iteration with a fleet of 100 000 vehicles. Maciejewski and Bischoff [105] used fleets of up to 11 000 vehicles in another study in the Berlin area, and for the largest fleet size more than 30 hours of runtime required to converge the scenario. Levin et al. [106] noted that in the case study with coordinated fleets in Austin, Texas (USA), only a sub-region was used due to performance limitations. Hörl et al. [107] reported that up to 4 hours was required to run a single daily iteration of a scenario for the city of Zurich (Switzerland) with a coordinated taxi fleet of up to 18 000 vehicles. The literature shows that large-scale multi-modal scenarios, which include public transit and coordinated fleets, typically

require long runtimes, and researchers tend either to reduce the complexity of the models or to use a sample of the population with downscaled infrastructures. Only a limited number of works use high-fidelity traffic simulation models for large-scale multi-modal scenarios with coordinated fleets and public transit systems, and, to the best of the author's knowledge, such scenarios were not demonstrated running on a GPU.

While many researchers tend to use downscaled scenarios with samples of populations to improve the runtimes, the consequences of such downscaling are not well studied. Moreover, there is evidence that small population samples can lead to discrepancies in traffic simulations. Ben-Dor et al. [108] reported issues when using a 10% population sample in MATSim simulations of the Tel Aviv metropolitan area. In these simulations, buses tended to get stuck in long waiting queues when the flow capacity of network links is over-used by private cars. Bischoff and Maciejewski [109] showed that a 10% population sample leads to 11% of the demand for autonomous taxis compared to a simulation with the full population in the case study in Berlin. At the same time, the deviations in the durations of trips with taxi vehicles involved were no more than 3%. Simoni et al. [110] noted that depending of the population sample size, the results of a MATSim simulation of central Zurich do vary. For example, flows on links decreased faster with the increased population sample size. Erath et al. [111] faced issues with overcrowded buses when simulating public transit in Singapore with a 10% population sample using MATSim. Bösch et al. [112] noted that in the Switzerland baseline scenario for MATSim one should not use population samples less than 5%–10% for scenarios with shared cars to prevent inconsistencies in the supply-demand balance. Kwak et al. [113] systematically studied the errors that arise in traffic simulations due to the use of samples of the full population with a macroscale static traffic assignment model. The work showed that the use of samples of the full population affects the predicted traffic flows even at the macro-scale. Llorca and Moeckel [114] studied the effects of downscaled populations in the scenario of the Munich metropolitan area. The authors demonstrated that the average travel time varies with the size of the population sample, with the minimum for a sample 10%–20%. The work showed that a scale factor of 5% can be used in simulations where only highly aggregated results are acceptable. Ben-Dor et al. [115, 116] focused on the impacts of downscaled scenarios in relation to car traffic using scenarios of a relatively small scale and providing subjective metrics. The authors emphasized the need of population samples of at least 25%–30% when disaggregated outputs are to be analysed. In overall, there is the lack of systematic and quantitative studies of the impacts of downscaled populations output results in multi-modal scenarios, including car traffic, public transit and coordinated fleets.

Finally, visualization and analytics of large-scale traffic simulations also pose challenges due to high volumes of generated data. Sewall et al. [117] presented an approach to visualize large-scale traffic flows on highways by presenting a lane as a continuum flow with a low variation in speed. This method, while not being an agent-based traffic simulation, demonstrated that a traffic model can be defined

to optimize visualization. Later, Sewall et al. [118] presented another method to visualize massive traffic flows reconstructed from spatio-temporal discrete data sources. Another approach implemented by Sewall et al. [119] uses a hybrid microscopic agent-based model which simulates and visualizes only the regions of interest, while the rest is modelled with continuum flows. This model achieved eight rendered frames per second with 190 000 vehicles simulated. Suzumura et al. [120] developed a large-scale traffic simulation platform that was able to run a Japanese nationwide daily simulation. While the authors mention that the platform has visualization capabilities for moving vehicles, no performance metrics or implementation details were provided. Shen and Jin [121] developed an agent-based system for detailed traffic animation in an urban environment using a car-following model with multiple driving profiles. A straight four-lane road with 40 000 vehicles took about 40 milliseconds to calculate a frame, and 20 milliseconds per frame if the lane-changing model is disabled. Heywood et al. [90] used the FLAME GPU [91] to run a microscopic traffic model on a GPU and visualize it in real-time. This is one of few works where a GPU-accelerated traffic model has been coupled with visualization. The results showed that it takes 1.2 milliseconds per simulated iteration to visualize vehicles, and that the visualization negatively impacts the simulation performance. Lu et al. [122] developed the Toolbox for Urban Mobility Simulations (TUMS) using LandScan [123] and microscopic agent-based traffic simulator TRANSIMS [101] to make it applicable globally. TUMS includes a macroscopic tool for link-based analysis in 15-minute intervals, and a microscopic tool for analysis of individual vehicles which are rendered with up to 1-second resolution. However, no information about the rendering performance of the tools was provided. The MATSim framework [76] for agent-based transport simulations has multiple tools to visualize and analyse outputs from scenarios. OTFVis has been developed mainly for the debugging of input data and provides limited visualization functionality. Another alternative is a commercial tool Via which provides better visualization capabilities, but the free version is limited in scale and functionality, and no runtime benchmarks for large-scale scenarios are available. Gehlot et al. [124] developed an agent-based evacuation simulator for large-scale scenarios with a microscopic traffic model. The visualization module was only evaluated qualitatively and is found to be smooth and efficient with a network of 4 000 nodes and 8 000 links and with up to 100 000 vehicles simulated. Kim et al. [125] developed a mesoscopic traffic simulator with a real-time visualization library, SALT-Viz. The rendering performance was evaluated using the Gangdong district's model in Seoul (South Korea) and the visualization part was able to output 148 frames per second. However, dynamically moving vehicles were not rendered, but rather network segments were colourised based on flow densities. Charlton and Laudan [126] presented a web-based platform to visualize outputs of MATSim in a web-browser. However, the performance of the presented platform is unclear as no specific runtime benchmarks were reported. To conclude, most of the prior works are focused on static and aggregated forms of visualization like plots, charts or distributions. Some works, that are focused on dynamic visualizations, typically use small-scale cases without quantitative

scalability benchmarks and without almost any technical details how the outputs of agent-based simulations are coupled with rendering hardware.

1.6 RESEARCH OBJECTIVES

Based on the provided literature review, the research objectives of the current work are the following:

- Design and implement a high-resolution traffic simulation model capable of running large-scale scenarios with millions of agents in a few minutes.
- Adapt the model to run simulations transparently on heterogeneous hardware including many-core CPUs and GPUs.
- Implement in the model existing and emerging transport modes other than cars, including public transit and coordinated fleets with detailed modelling of BEV technology, making it possible to run multi-modal scenarios.
- Using the developed framework, study the impacts of the deployment of coordinated fleets, both standalone and integrated with public transit systems, on transport systems and people's behaviour.
- Using the developed framework, identify and quantify the uncertainties of results coming from downscaled scenarios, including cars, public transit and coordinated fleets.
- Extend the simulation framework with visual analytics to simplify the development and post-processing of large-scale scenarios.
- Make the whole developed framework scalable and globally transferable, ready-to-use as an integrated solution for large-scale mobility simulations.

1.7 OUTLINE

The structure of the thesis is organized as follows:

Chapter 1 discusses the motivation for and topic of this study, identifying objectives which are covered in the consequent chapters.

Chapter 2 provides a short introduction to the concepts and challenges of GPU programming with a focus on the aspects that are tackled in the thesis. The overview of the main loop of the developed mobility simulation framework is presented. While the main focus of the thesis is the simulation of traffic and people's interaction with transport systems, a brief description of each part of the simulation loop is provided. A detailed methodology to simulate moving vehicles is then presented, including a queueing traffic model, gridlock

resolution mechanism and a velocity model to improve traffic dynamics for BEVs. The performance and scalability of the implemented loop is extensively benchmarked and profiled. How the developed framework can be adapted to support massively parallel heterogeneous CPU-GPU hardware with little effort is also demonstrated.

Chapter 3 describes the multi-modal extensions of the developed framework. These extensions include the simulation of an approximated flyover mode, public transit systems and coordinated fleets. Detailed descriptions of algorithms and data structures are provided for each of the implemented transport modes. The simulation framework is evaluated for performance and scalability with multi-modal scenarios. The public transit extension uses a novel modelling approach based on state machines in order to optimize execution on GPUs. For the fleet modelling, a dedicated simulation loop is presented together with scheduling algorithms and mechanisms for host-device synchronization. The developed fleet model is applied to study the impacts of the deployment of coordinated fleets in Zurich to replace private cars.

Chapter 4 describes a unified modelling pipeline designed to simplify and automate the generation of agent-based synthetic travel demand for large-scale scenarios. The pipeline takes available input datasets and, by performing a series of data transformations, generates disaggregated demand for the given area. Using Switzerland as a case study, it is demonstrated, in a step-by-step manner, how local mobility microcensus and other open datasets are used to build relevant discrete choice models (DCMs), which then are applied to the existing synthetic population of Switzerland to generate travel demand. It is demonstrated how the presented pipeline can be adapted to model the behaviour of people during the COVID-19 pandemic in Switzerland.

Chapter 5 presents the results from a case study in the Munich metropolitan region, where the impacts of integrated MaaS services composed of public transit and coordinated fleets are evaluated on a large scale. A modelling pipeline for Bavaria is built based on a German mobility microcensus in order to generate travel demand for the local synthetic population. A new transport mode, MaaS, is implemented in the simulation framework using intermodal routing with public transit as a backbone mode and a coordinated fleet feeding passengers at the first and last mile stages of their trips. Various fleet sizing options, together with distance limitation policies, are evaluated to study the potential benefits of such integrated systems.

Chapter 6 presents the results of applying the developed framework to evaluate the uncertainties in agent-based mobility simulations with downscaled inputs. First, existing measures of goodness-of-fit are evaluated to identify applicability for comparison of traffic and vehicle occupancy dynamics from two simulations. Second, in the absence of a statistically reliable measure, a

novel similarity measure based on the chi-squared test is introduced. Using previously developed scenarios for Switzerland and Munich, the uncertainties of simulation results using downscaled inputs are assessed for both disaggregated and aggregated variables. The analysis of uncertainties is performed for car traffic, occupancy of public transit vehicles and the predicted performance of coordinated fleets.

Chapter 7 demonstrates the capabilities of the visual analytics framework developed as a part of the integrated mobility simulation framework. The approach used to generate and store detailed events from simulated scenarios is presented. In contrast to other, event-driven simulators, these events are only used for analytical and visualization purposes, and can be generated and recorded upon a request. The architecture of the visual analytics framework is described, with graphical examples of post-processed outputs. The runtime performance and scalability of the developed framework are evaluated using large-scale scenarios of Switzerland and Hokkaido (Japan).

Chapter 8 summarises the main findings of this work and the outcomes of the case studies. It also suggests directions for future research.

2

MOBILITY SIMULATOR

All roads lead to people.
— Antoine de Saint-Exupery

The chapter is based on contributions from the following publications:

Saprykin, A., Chokani, N. & Abhari, R. S. GEMSim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios. *Simulation Modelling Practice and Theory* **94**, 199 (2019)

Saprykin, A., Chokani, N. & Abhari, R. S. *Large-scale multi-agent mobility simulations on a GPU: towards high performance and scalability* in. **151** (Elsevier, 2019), 733

Saprykin, A., Chokani, N. & Abhari, R. S. *Gridlock resolution in a GPU-accelerated traffic queue model* in. **170** (Elsevier, 2020), 681

Saprykin, A., Chokani, N. & Abhari, R. S. *A data-driven approach to run agent-based multi-modal traffic simulations on heterogeneous CPU-GPU hardware* in. **184** (Elsevier, 2021), 720

Plagowski, P., **Saprykin**, A., Chokani, N. & Shokrollah-Abhari, R. Impact of electric vehicle charging–An agent-based approach. *IET Generation, Transmission & Distribution* **15**, 2605 (2021)

This chapter, first, provides a short introduction into the concepts and challenges of GPU programming. Then, an overview of the developed mobility simulation framework is presented; detailed descriptions of the algorithms and data structures used to run GPU-accelerated mobility models are provided. The framework also includes gridlock resolution and velocity tracking models used to improve simulated traffic dynamics. In addition to agents with their logic, the state of vehicle agents is modelled separately. Finally, a hardware abstraction layer that allows massively parallel mobility models to be run transparently for users not only on GPUs, but also on many-core CPUs, is presented. The chapter aims to contribute to the field of GPU-accelerated multi-modal mobility modelling by implementing a full simulation loop with adaptive behaviour of the agents at a large scale with high temporal and spatial resolutions. Runtime benchmarks prove that the framework is scalable and significantly outperforms state-of-the-art tools that have similar functionality.

2.1 GPU BACKGROUND

The hardware design of GPUs is different from CPUs, which reflects different programming models. The basic knowledge of GPU hardware design is essential to understand the challenges of implementing agent-based traffic simulations and how the proposed solutions do work. While this work refers to GPUs and the CUDA framework developed by Nvidia, the same or very similar concepts are employed by other GPU vendors like AMD.

2.1.1 *GPU-based approach*

CPUs and GPUs have been designed to achieve different goals, and their architectures support these design goals. Figure 2.1 presents the schematics of the layouts of silicon dies for CPU and GPU types of devices. A typical modern CPU has up to several dozen cores, each of them decoding and executing program instructions in a parallel thread. A CPU core is designed to execute a stream (thread) of instructions as quickly as possible. It therefore incorporates quite complex data caching and flow control, which takes most of the transistors on a silicon die. In contrast, GPUs were designed to execute thousands of threads in parallel to efficiently render images from millions of vertices; hence, each core is executing a simpler logic in general. This difference results in hardware design: while CPUs have large caches with more complex memory hierarchies, GPUs use more transistors on a die to build data processing cores. To be more precise, GPU cores are not exactly the same cores as of CPUs, and one should have called them scalar processors or units, but during the course of this thesis both naming approaches are accepted and used interchangeably.

By providing more cores with data processing logic, a GPU can hide long memory access latencies with computation. That is, when a group of GPU threads is waiting for data from memory, another group of threads can use hardware resources to perform computations. A specialized hardware design allows GPUs to achieve the much higher instruction and memory bandwidth overall compared to CPUs. One can also look at the difference in the following way: CPU cores are much faster but have a relatively narrow computing frontier, while slower GPU cores have a much wider computing frontier. Depending on the workload type, the latter hardware design, when executed on a large number of cores, can achieve higher computing performance. However, in order to efficiently run code on a GPU, a different approach in programming is required.

The main challenge in developing software for GPUs is determining how to utilize the massively parallel architecture in the most efficient way. In many cases, this requires that existing software is re-written using different algorithms and data structures. Generally speaking, a distinct programming mindset is needed to design and implement software for GPUs, demanding more effort from a developer compared to the writing of software for execution on CPUs that have well-established

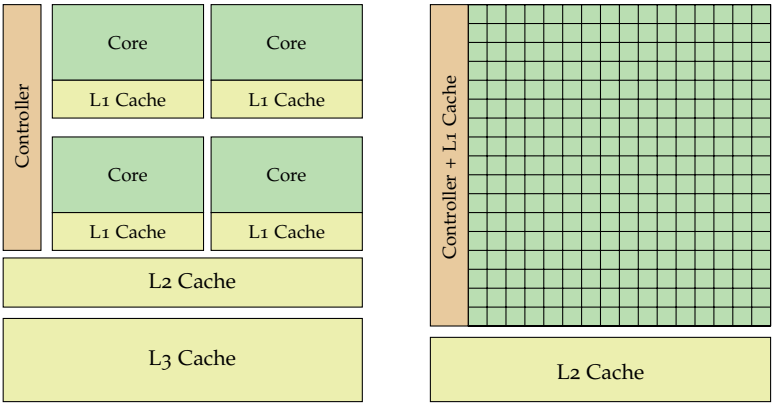


FIGURE 2.1: Schematics of silicon dies for CPU (left) and GPU (right). Computing cores are filled with dark green.

frameworks and methodology. Essentially, in light of recent advances in rapid application development, writing GPU-accelerated code is a step back that requires that a developer has a deeper understanding of hardware and, consequently, commits more time and effort to design software for it. The complexity of software development for GPUs is one of the main disadvantages of the technology; however, the situation is changing with many ready-to-use GPU-accelerated libraries and frameworks. For example, the CUDA framework from Nvidia allows one to develop software for massively parallel execution on a GPU using supplied libraries for ML, fast Fourier transforms, network communications and more. The future development of a GPU-accelerated software ecosystem is a key enabler for wider acceptance and use of this technology.

One should also note that not all software can be executed efficiently on a GPU. First, typical software represents a mix of sequential and parallelized code. When the sequential portion of the code dominates, and it is not possible to make it parallel, there is a little incentive to bring such software to GPUs as benefits would be negligible. Second, some tasks are simply not well suited for execution on GPUs. For example, a task may not provide sufficient parallelization, making it challenging to saturate GPU cores with enough work to hide memory latency; in this case, poor occupancy of GPU resources may lead to the opposite result when the execution of the code on a GPU takes longer than on a CPU. A deep domain knowledge of the problem to be solved on a GPU is another disadvantage of the technology, as the required assessment of the potential benefits of implementing software for GPUs has to be made in advance, in order to reduce the risks of wasted development resources.

That said, a GPU-accelerated model carefully split into parts executed separately on CPUs and GPUs, and the ways to effectively interact between these parts, shall

be considered. One can also see a GPU-accelerated model as a combination of hardware and software (programming) models, where the programming model defines and controls how the hardware part executes the code. Schematics of the CUDA software model and the GPU hardware model are presented in Figure 2.2 and described below in more detail.

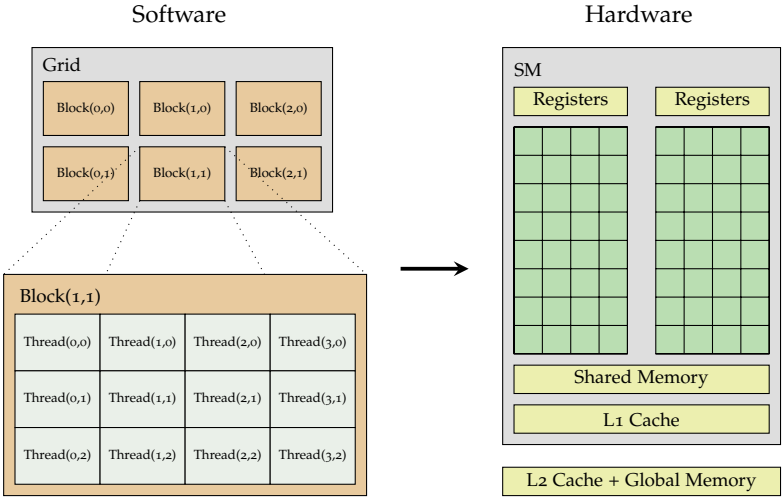


FIGURE 2.2: Software and hardware models of GPUs in Nvidia's CUDA representation.

2.1.2 *Hardware model*

On the hardware side, a GPU is composed of symmetric multiprocessors (SM). The smallest execution unit is a CUDA core (or scalar processor unit), and a typical SM contains from dozens to hundreds of cores. Each time an SM receives a block of threads, it splits them into groups and schedules the execution. A core executes a thread from the software side, and an SM schedules execution of threads on the available cores in groups of 32 called warps. In addition to scheduling, an SM also creates and manages the execution of the threads: when a block is finished, another waiting group of threads is scheduled to use the vacant resources. According to the SIMD (single instruction, multiple data) principle, all threads in a warp execute the same instruction to acquire performance gain. Individual threads in a warp always start at the same program address, but they have their own instruction address counter and a register state. This organisation of resources allows threads to diverge at branches and execute independently.

Branching the code requires special care when programming for GPUs. When a warp executes an instruction, full efficiency is achieved if each thread follows the

same execution path, otherwise the execution is serialized. If some threads in a warp diverge on their execution path because of a branching condition, then other threads are disabled until the diverged threads finish the execution. For example, if a code contains an *if-else* condition and half of a warp evaluates it to *true*, then, first, half of the warp is executed while the rest of the threads are disabled; afterwards, when the first half is finished, the second half of the warp is executed while the first one is waiting. Basically, the total execution time of the warp equals the sum of times required to execute each of the conditional branches by the halves of the warp, which means that each thread actually executes both branches. In other words, the execution is serialized. This is something that a developer should want to avoid in the models. To improve branching performance, GPUs exploit a SIMT (single instruction, multiple threads) approach similar to SIMD, but each thread of a SIMT block has its own stack pointer and can operate on a different data set. This allows for a GPU to automatically handle code branching through masking, while with the SIMD approach, a developer has to handle branching manually. Modern GPUs go even further and introduce so-called independent thread scheduling, when the GPU can group threads from a warp into SIMT units, hence allowing threads to diverge and reconverge at sub-warp granularity. When threads are scheduled independently within SIMT units, both code branches will be executed sooner or later, but still only a single instruction can be executed by a warp at a time.

An SM as a hardware unit has a limited amount of available resources like registers, parallel data cache or shared memory. These resources are partitioned among the executed warps, and the SM keeps the execution context for each warp until its end of life, hence, warps cannot migrate between SMs. On the one hand, this approach allows switching between execution contexts at no cost compared to expensive context switch of a CPU, and the warp scheduler can select a warp with the threads ready to execute (that is, when required data fetched or stored in memory, or arithmetic units of the SM are not used) at the time the instruction is issued. On the other hand, the number of created and managed blocks of threads on the SM depends on the code executed by threads. For example, if the code uses too many registers, then only a few blocks can reside on the SM, potentially making poor use of other hardware resources and reducing the overall runtime performance.

It would be also interesting to note that many modern CPUs, as they are reaching the limits of single-core performance, follow the trend for vectorized computations by implementing special blocks of SIMD units, and making them wider as well. This means that software designed for GPU execution may benefit later from running on such SIMD-accelerated CPUs as they become technically closer to GPUs.

2.1.3 *Programming model*

On the software side, threads are grouped into blocks, and blocks are grouped into a grid. Grids and blocks define a thread hierarchy within the executed code on

the GPU. A GPU maps blocks of threads on SMs, and, like warps, a block cannot migrate from one SM to another. Threads within the block can synchronize their execution and share SM resources, which provides additional opportunities for optimizations. A developer defines a function (kernel) that is executed by each thread on a GPU, and CUDA provides an indexing mechanism for threads within the kernel by assigning a unique thread identifier (ID) which is accessible through the built-in kernel variables. A thread ID can be calculated through the indices of a thread and the block in which it resides. Based on the indexing, threads can split the workload and communicate to each other. Consider the following example denoted in Listing 2.1, which adds, element-wise, two vectors of the same size and writes output to another vector.

```
__global__ void vector_add (float* in_a, float* in_b, float* out)
{
    int idx = threadIdx.x;
    out[idx] = in_a[idx] + in_b[idx];
}
```

Listing 2.1: Exemplary CUDA kernel for element-wise vector summation.

First, the keyword `__global__` indicates to the CUDA compiler that the code for this function shall be generated for both the host and the device. Second, each GPU thread uses a built-in three-dimensional (x, y, z)-variable `threadIdx` to get a thread index and split the workload. After the index is constructed, the sum of two elements from input arrays `in_a` and `in_b` is calculated pair-wise and is written into the output vector `out`. In general, a thread index is a three-dimensional vector, but in this example `y` and `z` components are ignored as the kernel is executed in a single one-dimensional block of threads, and the component `x` defines the thread ID. However, in real applications, a block may have multiple dimensions, and, moreover, multiple blocks can be also organized in a three-dimensional grid. The built-in kernel variable `blockDim` provides the size of a block in each dimension, and the built-in kernel variable `blockIdx` provides a block index within the grid, exactly in the same way that the variable `threadIdx` does. Additionally, the `gridDim` variable provides the size of the grid in each dimension. The presented thread hierarchy provides a natural way to partition a compound task in an array, matrix or volume. In a simple case, a thread ID in the grid of a single 3D block can be calculated as follows:

$$I_{1\_3D} = x + D_x \cdot y + D_x \cdot D_y \cdot z \tag{2.1}$$

Here, block dimensions are represented by the vector (D_x, D_y, D_z). When a grid consists of multiple 3D blocks aligned sequentially, a block index (x_b, y_b, z_b) within the grid is required to calculate a thread ID:

$$I_{1D\_3D} = I_{1\_3D} + D_x \cdot D_y \cdot D_z \cdot x_b \tag{2.2}$$

In the most complicated case, when a 3D grid consists of 3D blocks, a thread ID is calculated in two steps. First, a block ID within the grid of dimensions $(D_{g,x}, D_{g,y}, D_{g,z})$ is calculated:

$$I_{b,1D\_3D} = x_b + D_{g,x} \cdot y_b + D_{g,x} \cdot D_{g,y} \cdot z_b \tag{2.3}$$

Then, the final thread ID can be obtained as follows:

$$I_{3D\_3D} = I_{b,1D\_3D} \cdot D_x \cdot D_y \cdot D_z + D_x \cdot D_y \cdot z + D_x \cdot y + x \tag{2.4}$$

Listing 2.2 shows a code snippet that allocates memory for data arrays on both the device and host sides, transfers required data to the device, calls the previously defined vector summation kernel from Listing 2.1, and copies the results back from the device to the host. One can note how grid and block dimensions are passed to the kernel at the launch time through a special chevron syntax (<<< >>>). Assuming that the size of vectors is N, the kernel can be launched with a single-dimension grid of size 1 (the first parameter within chevrons) and a single-dimension block of size N (the second parameter within chevrons).

```
size_t dsize = N * sizeof (float);

// Allocate memory on device (GPU side)
float *vec_a = (float *) cudaMalloc (dsize);
// Do similarly with vec_b and vec_out

// Allocate memory at host (CPU side)
float vec_a_host[N];
// Define similarly vec_b_host and vec_out_host

// Fill host arrays with data
// ...

// Copy data from host to GPU
cudaMemcpy (vec_a, vec_a_host, dsize, cudaMemcpyHostToDevice);
cudaMemcpy (vec_b, vec_b_host, dsize, cudaMemcpyHostToDevice);

// Run GPU code
vector_add<<<1, N>>> (vec_a, vec_b, vec_out);

// Copy results from GPU to host
cudaMemcpy (vec_out_host, vec_out, dsize, cudaMemcpyDeviceToHost);

// ... Do data processing and free device memory
cudaFree (vec_a);
// Do similarly for vec_b and vec_out
```

LISTING 2.2: A typical approach of memory management and execution of code on GPUs, with data transfer between the host and the device.

The above-mentioned thread indexing system provides a flexible way to split the workload, but a developer has to find a reasonable compromise between SM occupancy and the performance of a single block to maximize the overall performance.

As discussed before, large blocks may lead to poor resource usage, while smaller blocks can reduce the performance by wasting memory bandwidth. There are also some physical limitations of the grid and block dimensions as block hardware resources reside on the same SM until the block finishes its execution. This is one of many challenges arising during the development of GPU-accelerated models. Typically, a kernel grid is either picked manually by doing test runs, or is determined dynamically at runtime.

2.1.4 *Memory management*

One aspect of GPU programming that separates it from the CPU-based programming to which the vast majority of developers are used, is memory management, including allocation, deallocation and transfers between the host and the device. The reason for this is how the GPU programming model is organized. While kernels are executed on a physically separate device, the host continues running the program that controls the execution of the code on the device. This assumes that the device has its own memory spaces, physically separate from the host. As kernels work with device memory only, a host program has to manage device memory spaces visible to kernels; in the same way, to access data from the device, the host has to transfer it from the device first. Unified Memory technology, available from CUDA, allows bridging of the host and device memory spaces by providing automatically managed memory. Managed memory is a common, coherent address space available to all CPUs and GPUs installed in the system. While Unified Memory simplifies the memory management process, this work is focused on the most generic approach with manual control of memory management. The manual control of memory management not only allows for better optimization possibilities in given hardware resources, but also reduces performance overheads introduced by other memory management technologies. In Listing 2.2, CUDA functions `cudaMalloc` and `cudaFree` are used to allocate and free the device memory, respectively, and the CUDA function `cudaMemcpy` is used to move data between the host and the device. Depending on the executed software model, data transfers could be one of the major bottlenecks in the runtime performance of GPU-accelerated code.

Another challenging aspect of GPU programming is the optimization of memory access. During the execution, a GPU thread has access to multiple memory spaces, specifically:

- **Thread-local** memory, which is accessible only by the thread, making this memory private. Depending on the data size of local variables, it can be placed either in registers (low latency) or in global memory (high latency).

- **Shared (on-chip)** memory is visible only to the threads of a block, and this memory can be used to implement within-block optimizations. Shared memory has a relatively small size (kilobytes) and much lower latency than

global memory, but is organized in memory banks that can cause access conflicts with increased latency.

- **Global (off-chip, DRAM)** memory is accessible by all threads, but has very high latency and large size (gigabytes).

- **Constant** memory is accessible by all threads and it resides in the global memory space, but is cached in the constant cache. This memory has very low latency when threads hit the cache and jointly access the same address, making it suitable for storing constant data used by threads through the same addresses.

- **Texture and surface** memory is accessible by all threads and it resides in the global memory space, but is cached in the texture cache. This memory is designed for fetches with constant latency; even a cache hit does not reduce the latency. The texture cache is optimized for 2D spatial locality, hence, when threads from a warp read data that is close in 2D, optimal performance can be achieved.

By using the terms "low latency" and "high latency", one can expect that some memory spaces like shared memory are accessible in a few clock cycles, while others like global memory require a few hundred clock cycles. Typically, memory on GPUs is managed manually by a developer, who has to decide how to split data across multiple memory spaces, and which data structures and algorithms to use in order to maximize the performance. In general, the performance of a code is limited in the following ways:

- **memory bound** code is when the measured performance of the memory system is close to or at the maximum;

- **compute bound** code is when the measured throughout of compute instructions is close to or at the maximum;

- **latency bound** code is when it is neither memory bound nor compute bound, and typically happens when GPU hardware is not saturated with enough work.

As will be shown later, a GPU-accelerated part of the developed mobility simulator is mostly latency and memory bound as most of its runtime is spent in propagating agents through the network graph using spatial queues. Moving agents (persons and vehicles) between queues does not require extensive computations but rather the execution of many reading and writing transactions through global memory. While some computations are performed to estimate vehicle dynamics or track the state of charge (SoC) of BEVs, the number of computations is negligible compared to the number of global memory transactions. In order to improve access to the global memory, all threads within a warp shall access memory in a certain continuous physical range when executing an instruction (this is referred to as coalesced access).

When threads within a warp access memory in a scattered manner, the memory latency increases substantially.

Coalesced access can be achieved by using optimized data structures in GPU memory [128, 129]. There are two main approaches to storing arrays of items: array of structures (AoS) and structure of arrays (SoA). The difference between these approaches is shown in Listing 2.3: when a thread with index i in a warp processes the `x` coordinate of the object with index i in the array, then with AoS (lines 3–9) the data is scattered with a gap of `sizeof(point3_aos)` for any threads i and $i+1$ in the warp. In contrast, with the SoA approach (lines 11–17), the data for the warp is organized in one single and continuous chunk of memory. The former requires more memory bandwidth to execute an instruction for the warp, thus reducing the overall performance.

```
1  #define NPOINTS 32
2
3  // Array of structures
4  struct point3_aos {
5      double x;
6      double y;
7      double z;
8  };
9  point3_aos points_aos[NPOINTS];
10
11 // Structure of arrays
12 struct point3_soa {
13     double x[NPOINTS];
14     double y[NPOINTS];
15     double z[NPOINTS];
16 };
17 point3_soa points_soa;
```

LISTING 2.3: Array of structures (AoS) and structure of arrays (SoA) approaches to store 3D points.

The above example, showing a warp of threads trying to access the `x` coordinate of the points using structures in Listing 2.3, with each thread reading its own point, is represented in another way in Figure 2.3. When AoS is used, data points are allocated sequentially in memory, one after another, making gaps between the same coordinates of close-by points. With SoA, `x` coordinates of the points are allocated sequentially in memory first, followed by the arrays of `y` and `z` coordinates. In the former case, two adjacent threads from the warp will have a gap between memory transactions of 24 bytes, while in the latter case there would be no gaps. This means that with the AoS approach only one third of the memory bandwidth is utilized while the rest is wasted. In contrast, the SoA approach does not waste memory bandwidth at all and the code executes faster.

Obviously, coalesced memory access is not always possible, especially when using data structures with random memory access patterns like trees or linked lists. For

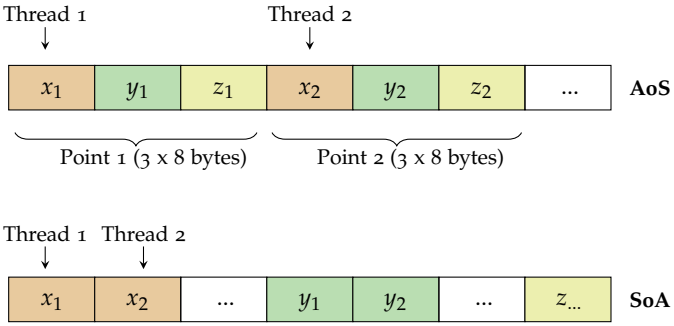


FIGURE 2.3: Scattered (top) and coalesced (bottom) memory access patterns of threads within a warp on a GPU.

such situations, there are other potential optimizations available for GPUs. For example, if a kernel executes a lot of scattered memory transactions, one could try to disable (bypass) the L1 cache to avoid using wide transactions that waste the memory bandwidth. The reason for such behaviour by the L1 cache is as follows. Typically, GPUs can execute 32- to 128-byte-long memory transactions. However, when the L1 cache is turned on, a device will stick to 128-byte transactions as it fits the size of a cache line, wasting a large fraction of the bandwidth. When the L1 cache is turned off, the device can use shorter, 32-byte transactions and waste less of the bandwidth. This behaviour of caches, however, varies depending on the capabilities of a GPU, and some devices do not promote by default a full cache line of 128 bytes, but operate with 32-byte long sectors of a cache line, and only fetch missing sectors of the accessed cache line.

Another issue that may arise during the development of GPU-accelerated agent-based models is memory alignment and the related alignment of data structures. For example, Nvidia GPUs allocate memory with the alignment of at least 256 bytes, which means that even if a program needs to allocate four bytes, it will eventually use 256 bytes of the device memory. Having millions of agents and their individual data to simulate, a common approach for CPU-based programming when the memory is allocated in per-agent chunks, may lead to excessive use of memory when applied to GPUs. In such circumstances, one must effectively implement its own memory allocator to distribute GPU memory to store agent-based data more efficiently.

The optimization of global memory access through efficient data structures and algorithms, use of higher levels of the memory hierarchy like the L2 cache, and the use of diverse memory spaces are among the major focuses of this thesis. Thus, the above-mentioned memory access optimizations, as well as others, will be introduced later in the context of mobility simulations.

2.2 FRAMEWORK OVERVIEW

During the course of the thesis, the GPU-enhanced mobility simulator (GEMSim) was developed as part of EnerPol, a bottom-up, integrated simulation framework for scenario-based analysis of energy, urban planning and population dynamics [130–132]. GEMSim integrates into EnerPol the capability to assess mobility. C/C++ programming languages were used to efficiently work with hardware and to have maximum control for performance optimizations. Nvidia's CUDA SDK for GPU programming was used for GPU acceleration.

GEMSim's main simulation loop implements a co-evolutionary [133] learning process for the agents. The structure of the main loop is shown in Figure 2.4 and its parts are described later. The loop iteratively performs heterogeneous computations that are shared between the host and the GPU, until it converges a scenario to a Nash equilibrium [134] state, when an agent cannot improve their daily performance unilaterally. Wardrop [135] suggests that this state can be used to predict traffic patterns in transport networks.

One of the fundamental problems that may arise when switching from a CPU-based mobility simulator to a GPU-based simulator is how to handle simulation events. Many existing simulators rely on the events generated during a simulation in order to improve performance and provide more flexibility for extensions. An event occurs when something happens (that is, an agent enters a link), and a simulator needs to react only to generated events, rather than checking at each time step for all possible changes from the previous step. For multiple reasons, GPUs are not well suited for such asynchronous events. First, it is difficult to process events in a coalesced manner because a typical event has references to agents, links, vehicles and other objects. Second, some events require a corresponding match with another event (for example, an agent entering and leaving a link) and with a limited dynamic memory allocation and limited memory capacity on a GPU device, the processing of events may be impractical, especially for large-scale scenarios.

GEMSim's simulation loop does not rely on events, and all necessary reactions are incorporated into the GPU code directly within the pre-allocated device memory. For example, congestion is calculated in-situ when each agent passes a link from his/her route. However, events can be generated and recorded during the simulation for further post-processing and analysis.

A scenario configuration file contains all the information required for GEMSim to run the simulation loop, including input datasets and other parameters. The simulator reads a configuration file, then reads specified input datasets like network and plans, pre-processes the data if necessary, and executes the simulation loop according to the provided parameters.

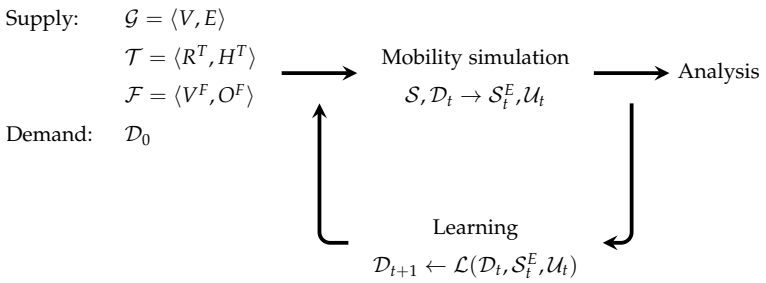


FIGURE 2.4: Structure of the GEMSim's main co-evolution simulation loop.

2.2.1 *Supply and demand*

The supply part $\mathcal{S}$ of the input data is represented by a multi-modal network $\mathcal{G}$, and optional inputs like public transit schedule $\mathcal{T}$ and fleet operators $\mathcal{F}$. The network $\mathcal{G}$ is a directed graph with a set of links E and nodes V, where a node represents an intersection of roads, and a link is a road segment between two nodes. Each link is described with the physical properties that are required for the traffic simulation model: length, number of lanes, flow capacity, free speed, and the set of allowable transport modes.

The public transit schedule, $\mathcal{T}$, is composed of a set of transit routes R^T and a set of transit stops H^T. Each transit route defines a network on $\mathcal{G}$, an ordered set of transit stops from H^T to follow; and departure frequencies for the initial terminal stop of a route.

Each of the fleet operators from $\mathcal{F}$ is composed of a set of vehicles from V^F and a set of operating policies O^F. Fleets of vehicles can either be generated automatically using a default type if their performance does not matter during the simulation (i.e., no need to track SoC of BEVs) or can be provided explicitly. Operating policies define how a fleet operator works and reacts to the events, for example, how long to perform pickup and drop-off, or which algorithms to use for demand-supply optimization.

The demand part, $\mathcal{D}$, of the input data is a population of agents each of whom has its own set of daily plans, P. Each plan $P_k \in P$ is defined as a set, A_k, of spatially distributed activities of different type and duration, and a set, L_k, of legs with routes to travel between activities. Agents only switch transport modes at places of activities; a special class of *merge* activities is defined in order to delay an agent for a specified amount of time, such that a line transfer in public transit, or a switch from walking to public transit, can take place.

A simulation is performed in an iterative manner. After the initial demand $\mathcal{D}_0$ has been assigned for the first iteration, the mobility simulator performs network loading based on the individual daily plans of the agents. Only one plan P_k per agent

is selected for execution on the t-th iteration. Agents perform their activities and travel between locations, therefore generating traffic externalities such as congestion on the network $\mathcal{G}$ or occupancy of the transit routes R^T. The simulated duration is typically somewhat longer than a day to let agents who depart late reach their final destinations. The main outputs from the mobility simulator in the t-th iteration are the externalities $\mathcal{S}_t^E$ related to input supply and the scores $\mathcal{U}_t$ of the plans. A learning process, $\mathcal{L}$, is then executed with $\mathcal{D}_t$, $\mathcal{S}_t^E$, and $\mathcal{U}_t$ as the input data, and a new demand $\mathcal{D}_{t+1}$ is output. This newly obtained demand $\mathcal{D}_{t+1}$ is used in the next $t+1$ iteration of the simulation process. After performing T iterations the analysis is executed and the simulation is finished.

2.2.2 *Scoring*

A scoring function for the plans can be implemented in any form, but usually an agent gets a positive score when performing activities and a negative score when travelling, waiting or upon late arrival at a place of activity. Currently, the Charypar-Nagel [136] scoring (utility) function formulated from the Vickrey model [137] for road congestion is implemented:

$$U(P_k) = \sum_{a_i \in A_k} U_{act}(a_i) + \sum_{l_i \in L_k} U_{trav}(l_i) \tag{2.5}$$

Here, the function $U(P_k)$ estimates the score of a plan P_k by summing up scores of each activity using the function U_{act} and scores of each travel leg using the function U_{trav}. The utility of an activity U_{act} is defined as follows:

$$U_{act}(a_i) = U_{dur,i} + U_{wait,i} + U_{late,i} + U_{early,i} + U_{short,i} \tag{2.6}$$

$U_{dur,i}$ is the utility of performing an activity for a certain duration t_{dur} and is defined as follows:

$$U_{dur,i}(t_{dur}) = \beta_{dur} \cdot t_{typ} \cdot ln\left(\frac{t_{dur}}{t_0}\right) \tag{2.7}$$

where β_{dur} is the marginal utility of performing an activity, t_{typ} is the typical duration of the activity, and t_0 is the duration at which the utility start to be positive. As activities may have different priorities, the value of t_0 depends on the priority p:

$$t_0 = t_{typ} \cdot e^{-\frac{1}{p}} \tag{2.8}$$

When the activity duration t_{dur} is below or equal to the value of t_0, the utility $U_{dur,i}$ becomes negative and is calculated as follows:

$$U_{dur,i}(t_{dur}) = -(t_0 - t_{dur}) \cdot \frac{\beta_{dur} \cdot t_{typ}}{t_0} \tag{2.9}$$

$U_{wait,i}$ is the utility of waiting (that is, when the facility is closed) and is defined as follows:

$$U_{wait,i}(t_{wait}) = \beta_{wait} \cdot t_{wait} \tag{2.10}$$

where t_{wait} is the waiting time and β_{wait} is the marginal utility of waiting (typically, it is negative). $U_{late,i}$ is the utility of arriving at the place too late, and is defined as follows:

$$U_{late,i}(t_{start}) = \beta_{late} \cdot (t_{start} - t_{latest}) \tag{2.11}$$

where t_{start} is the start time of the activity, t_{latest} is the latest possible start time of this activity, and β_{late} is the marginal utility of being late (typically, it is negative). $U_{early,i}$ is the utility of finishing the activity too early, and is defined as follows:

$$U_{early,i}(t_{end}) = \beta_{early} \cdot (t_{earliest} - t_{end}) \tag{2.12}$$

where t_{end} is the end time of the activity, $t_{earliest}$ is the earliest possible end time of the activity, and β_{early} is the marginal utility of being early (typically, it is negative). $U_{short,i}$ is the utility of performing the activity for a too short time, and is defined as follows:

$$U_{short,i}(t_{dur}) = \beta_{short} \cdot (t_{shortest} - t_{dur}) \tag{2.13}$$

where $t_{shortest}$ is the shortest possible duration of this activity, and β_{short} is the marginal utility of staying too short (typically, it is negative). Utilities $U_{late,i}$, $U_{early,i}$ and $U_{short,i}$ are calculated only in case the respective violation happens, otherwise these utilities are set to zero. Finally, the utility (typically, it is negative) of a travel leg U_{trav} is defined as follows:

$$U_{trav}(l_i) = U_{trav}(t_{trav}, d_{trav}) = \beta_{ttime} \cdot t_{trav} + \beta_{tdist} \cdot d_{trav} \tag{2.14}$$

where t_{trav} is the travel time, d_{trav} is the travel distance, β_{ttime} is the marginal utility of travel time (typically, it is negative), and β_{tdist} is the marginal utility of travel distance (typically, it is negative or equal to zero).

Usually, agents improve their score $\mathcal{U}_t$ relatively quickly in a few iterations, but then the impact of the learning process decreases exponentially as the population reaches a Nash equilibrium.

While in other mobility simulators such as MATSim [76] the scoring procedure in the simulation loop typically relies on events, in GEMSim the scoring is executed on a GPU where required data for scoring is recorded during the simulation to avoid using events. For scoring, start and end times of travel legs and activities are required, as well as scoring parameters, and the plan data structure on a GPU contains the corresponding fields. At the end of an iteration, a scoring GPU kernel is executed and each of GPU threads calculates a score for one of the agents.

2.2.3 *Learning*

The learning process, $\mathcal{L}$, allows agents to adapt their daily plans from the previous iteration before starting the next iteration. A genetic algorithm was implemented, whereby a sample of the population is selected based on a given probability, and then each agent from the population sample selects one of the plans P_k from memory, and modifies or leaves the plan as is. A learning strategy comprises two components

(either of which can be omitted): (i) a plan selector and (ii) a plan modifier. A plan selector picks one of the plans from the agent's memory during the learning stage, while a plan modifier changes the selected plan. The following plan selectors were implemented in GEMSim:

- **best score** selector picks a plan with the highest score;
- **worst score** selector picks a plan with the lowest score;
- **random** selector picks one of the plans randomly;
- **keep** selector picks the same plan that was executed in the last iteration;
- **exp-beta** selector picks one of the plans at random, with the probability of the k-th plan being selected is defined as follows: $p_{i,k} = V_{i,k} / \sum_j V_{i,j}$, where $V_{i,k} = e^{(U_{i,k} - U_{i,best}) \cdot \beta_U}$, and $U_{i,k}$ is the score of the k-th plan of the current agent i, and $U_{i,best}$ is the best score in agent's memory of plans, and β_U is the relaxation parameter set to 1 by default.

In addition to plan selectors, the following plan modifiers were implemented:

- **re-routing** modifier estimates routes between the locations of the activities in the plan considering the congestion patterns from the last simulated iteration;
- **mode change** modifier switches the transport mode of the travel legs in a plan uniformly at random from the list of allowed modes;
- **time change** modifier shifts the start times of all activities in a plan by a time period sampled uniformly at random within the allowed time window.

In the scenario configuration, a set of used strategies is defined with the respective probabilities of being executed on the population sample chosen for behaviour adaptation between the iterations. By splitting learning strategies into selectors and modifiers, GEMSim provides a flexible mechanism to re-use already existing implementations in higher-level user-defined strategies. While the set of provided default plan selectors and modifiers might be considered rather limited, in practice, strategies constructed from these items converge scenarios in an efficient way. More sophisticated strategies can be implemented by users for specific needs, including DCMs and RL [138].

As discussed in the previous chapter, not all parts of the code can be efficiently parallelized, and not all the parts that can be parallelized are suitable for GPUs. Based on existing literature [139], there are two major bottlenecks in agent-based mobility simulations: traffic propagation and learning. While traffic propagation has great potential to improve runtime performance, the learning part is not that obvious. The most widely used learning strategy, and the most demanding for computing performance, is re-routing of agents based on congestion from the previous iteration. Typically, about 10% of the agents are re-routed between the iterations [140, 141]. This strategy essentially performs the relaxation of traffic between the iterations

and contributes substantially to the convergence of a scenario. The reason why the re-routing strategy takes lot of runtime is the scalability of routing algorithms.

In general, there are two types of routing algorithms for handling input data: with pre-processing and without (on-the-fly) [142]. While the former algorithms can provide a substantial speed-up in runtime, they require some time for data pre-processing, and this time can increase up to hours for large-scale road graphs. Considering that a typical scenario requires that hundreds of iterations are run, and after each of the iterations travel times across the road graph change due to the learning process, it is impractical to use routing algorithms with long pre-processing times. Moreover, not all of these algorithms account for dynamic weights as the travel time varies during the day. Routing algorithms without pre-processing are usually more flexible and have a weight function defined that can be implemented in any way. However, the disadvantage is that these algorithms have to explore a much wider search space to find an optimal solution, and the time may grow non-linearly with the size of the road graph.

To keep the re-routing strategy flexible, the decision was made to use algorithms without or with little (once per scenario run) pre-processing; most of them are based on the Dijkstra algorithm [143]. In recent years, GPU-accelerated graph algorithms for analytics and path finding have received attention in the literature [144–148] with impressive speed-up factors (an order of magnitude or more) achieved over CPU-based versions of the same algorithms. However, transport networks (and corresponding graphs) are different from the graphs used for runtime performance evaluation in the aforementioned studies. Unlike other, dense graphs, transport graphs have loosely connected nodes, and the average degree of a node (the number of edges that are incident to a node) varies from two to six when a uni-directed graph is used. Another aspect of transport graphs is that they are uni-directed, and each node has roughly half as many outgoing edges (as many roads are actually bi-directional). The Dijkstra algorithm works, in principle, by expanding the search area through the outgoing edges of the nodes, and many GPU-accelerated algorithms parallelize this expansion mechanism. This means that GPU-accelerated graph algorithms deliver better runtime performance with denser graphs, while for transport graphs in can be difficult to saturate a GPU with enough work to be done in parallel.

Table 2.1 shows runtimes for the Dijkstra-based implementation [147] of a single-source shortest path (SSSP) problem algorithm on both a CPU and a GPU applied to multiple transport graphs. The algorithm calculates the shortest paths in a graph from one of the source nodes to the rest of the nodes. The graphs used in runtime evaluation have been obtained from the OpenStreetMap (OSM) [149] service followed by a topology simplification procedure. The network simplification procedure removes nodes that are not essential for traffic simulation and routing; these are the nodes that define the curvature of the roads. After unused nodes are removed and other edges consolidated and updated, the performance of routing was measured. Tests were run using AMD EPYC 7742 CPU and Nvidia V100S GPU, the CPU-based version of the algorithm is single-threaded.

TABLE 2.1. Comparison of performance for Dijkstra routing algorithm on a CPU and a GPU.

Location	Nodes	Links	Average node deg.	Execution time, ms GPU	CPU
Switzerland	513 K	1 127 K	4.39	104.25	65.78
Bosnia and Herzegovina	176 K	400 K	4.55	70.64	18.98
Bavaria (Germany)	290 K	841 K	5.80	34.68	32.51
Los Angeles (USA)	460 K	1 214 K	5.28	34.20	51.66
San Diego (USA)	280 K	765 K	5.46	29.97	24.01
San Francisco (USA)	818 K	2 086 K	5.10	732.55	91.45
Hokkaido (Japan)	333 K	978 K	5.88	61.55	42.55

As one can see, the main finding is that there is no clear advantage to using a GPU over a CPU for the simple SSSP algorithm. For most of the cases, both the CPU and GPU showed comparable performance, and in some cases either the GPU or the CPU was a bit faster than another. The case of San Francisco is notable in that the GPU runs eight times slower compared to the CPU. The reason could be a relatively low average node degree, as well as peculiarities of the road graph. The Swiss road graph also provides poor GPU performance, although not at the same magnitude as in San Francisco. One can also consider that a more complex logic of routing like dynamic weights, which is actually required in a practical simulation framework, will probably lead to greater performance degradation on the GPU compared to the CPU. The reason is that code branching and fetching additional data from memory with high latency are more expensive to handle on GPUs.

Another practical aspect of the routing algorithms used in mobility simulations is that most of the time only a point-to-point route is required, when an agent needs to know how to drive from one location to another. This means that only a subset of a graph is explored, in contrast to the SSSP where the whole graph is visited by the algorithm. A routing algorithm visiting fewer nodes will lead to even less work that can be parallelized by a GPU, therefore the runtime performance of a GPU may even decrease compared to a CPU-based implementation.

To conclude these notes on routing, the experiments show that there is no clear advantage in using GPU-accelerated routing algorithms. Moreover, data transfers between the host and the device, which might be required for a more complicated routing logic, were not considered in this comparison. Therefore, GPUs do not currently provide a robust and superior solution to solve the routing problem for mobility simulations than CPU-based implementations; hence, the decision was made to keep this part of the simulation loop at the host side. However, as there is active research and development into GPU-accelerated routing for both software and hardware, the situation may be improved in the future.

Dijkstra and A* routing algorithms [64, 150] were implemented in GEMSim. The version of the A* algorithm includes an overdo factor and landmarks with multiple allocation strategies. These algorithms have also many-to-many versions to route from multiple sources to multiple sinks. The reason for selecting the A* algorithm is based on the nature of the learning process, when dynamic weight of congestion has to be accounted for at the road graph level. In this regard, A* algorithm only requires one to specify the lower bound of the travel cost between the points. The lower bound can be calculated as dividing the Euclidean distance by the fastest possible speed limit used in the network. The advantage over other algorithms is that no data processing is required between the iterations once initial landmarks are calculated.

Runtime performance of Dijkstra-based algorithms greatly depends on the implementation of the priority queue used to sort graph nodes by the cost it takes to reach them. GEMSim uses d-ary min-heap [151] with $d = 4$ and the heap being allocated and aligned [152] to a typical CPU cache line size of 64 bytes. Another typical performance bottleneck in routing algorithms with piece-wise cost function on the graph edges (that is, the cost changes each 15 minutes based on congestion evaluated previously) is data mapping for an edge and a time period. A common approach to solving this problem is to use an associative array (a key-value map) that connects congestion piece-wise data with a network link through the link ID. However, when using a router with a large-scale network with millions of links, millions of mappings from a link ID to actual data, used in the cost function, must be resolved, leading to long runtimes. Instead, index-based access to congestion data has near-constant time, and this approach was implemented in GEMSim where each network element, in addition to a unique ID, has an index associated. These indices, as will be showed later, can be used not only to speed-up the routing process, but also to store data on GPUs.

To utilize available CPU resources efficiently, a router can perform multi-threaded processing by splitting agents among multiple CPU cores. As will be demonstrated, a more efficient implementation of routing algorithms brings considerable improvement in runtime compared to other mobility simulators using the same routing algorithms. One should note that, while the learning part of the simulation loop is executed on the host side, nothing prevents users from implementing their own GPU-accelerated strategies, for example using AI and ML algorithms to model the behaviour of people. In fact, it has already been demonstrated [138] that GEMSim can be coupled with GPU-accelerated RL algorithms executed in the learning strategies.

2.2.4 *Outputs*

Some data that reside on a GPU at the end of the simulated iteration are required to run the next iteration, and some data are optional. Mandatory data must be downloaded from the GPU and post-processed before starting the next iteration.

An example of mandatory data is congestion statistics for each link in 15-minute intervals. These congestion statistics are used to re-route agents during the learning stage. Traffic counts for each link and agent state counters (departures, arrivals, en-route) are considered optional data. Other optional data include trip statistics per transport mode, plan execution statistics, score statistics, public transit occupancies and spatio-temporal statistics for each of the fleet operators. Additionally, a stream of detailed events from the simulation can be recorded. This stream includes every event that happened and can be used for custom post-processing, analysis and visualization. The mobility simulator allows flexible configuration of what data shall be recorded and how frequently, and the output datasets (which consume a considerable amount of disk space) are automatically compressed.

2.3 TRAFFIC PROPAGATION

The traffic propagation part of the simulation loop, which performs the actual mobility simulation, was accelerated with the GPU, thus removing one of the runtime performance bottlenecks (learning step being another bottleneck). This section of the thesis provides relevant background on the topic and a detailed description of the GPU-accelerated implementation of the traffic model.

2.3.1 *Background*

Over recent decades a number of agent-based mobility simulation tools have been developed, including: TRANSIMS [101], MATSim [76], SUMO [102], DRACULA [153] and DynaMIT [154]. In general, traffic simulations can be classified [155] as either:

- **microscopic**, which model in detail the dynamic of each vehicle and their interactions with others;
- **mesoscopic**, which aggregate vehicles in homogeneous groups and model interactions with little detail;
- **macroscopic**, which simulate aggregated traffic streams without distinguishing their constituent elements.

Some authors have also introduced another class of simulations, nanoscopic [156, 157], which separately model parts of vehicles, or human thinking and decision-making while driving. In this work, however, nanoscopic elements are considered part of the microscopic approach.

Macroscopic models have limited applicability for agent-based systems as such models aggregate individual behaviours of agents, and lose interaction dynamic between them. On the other hand, a microscopic model can perform simulations with the highest level of detail, although it does not come for free. One of the main disadvantages of microscopic models is the need for an extensive and time-consuming

calibration process, where each behavioural sub-model (i.e., car following, lane changing, etc.) is adjusted to fit the local context. Another disadvantage of the microscopic approach is that a highly-detailed infrastructure model is required in order to make these models more realistic and useful. For example, intersections need to be described with traffic lights and rules for lane turning. Such data might not always be available, and it often requires time to collect and prepare. Finally, microscopic models are more computationally demanding as the state of simulated agents is tracked individually in each time step. Mesoscopic models provide a balanced trade-off between the high runtime performance of macroscopic models and the high level of detail provided by microscopic models. In this work, a mesoscopic approach was used to implement dynamic traffic assignment. This allows one to efficiently simulate millions of agents, and, where necessary, enrich the model with the elements found in microscopic models.

As mentioned in Chapter 1, traditional approaches, used to speed up existing traffic models, do not bring the desired improvement in runtime performance. Therefore, one of the goals of the thesis was to utilize highly-parallel modern hardware in order to accelerate traffic simulations and make it possible to run large-scale scenarios with millions of agents in a few minutes. Though some attempts have been made to bring agent-based simulations onto GPUs and FPGAs, to date none of these attempts have succeeded in providing a ready-to-use product for large-scale and multi-modal scenarios.

The semiconductor industry in the HPC area is moving towards highly-parallel hardware solutions: many-core CPUs, GPUs, FPGAs and custom accelerators for AI training and inference are among the main trends. While GPUs have been designed to efficiently render millions of vertices in parallel from the beginning, in recent decades these devices have been adapted for general-purpose computing to process not only graphics, but other data in parallel. This gave rise to a new direction in HPC, and now GPUs are extensively used in many areas, especially in ML. Traffic simulation is not an exception, and many researchers have demonstrated the potential of GPU technology with promising results.

It should be noted that distributed computing introduces additional complexity into models, and also increases maintenance costs as expensive hardware is required. On the other hand, GPUs provide both a higher level of parallelism compared to CPUs, and avoid issues related to the complex distributed architecture and expensive hardware of distributed systems. Thus, several attempts have been made to accelerate ABMs using GPUs.

One early attempt to perform agent-based traffic simulations on a GPU was made by Strippgen and Nagel [75]. A functional queue-based mesoscopic model was developed and speed-up factors of 5.5 to 67 (compared to an optimized single-core Java version) were achieved. Though the speed-up was promising, the developers faced issues while integrating the prototype into MATSim [76]. Specifically, the achieved performance improvements were not reached in MATSim. Although the model itself was quite simple (that is, agents did not adjust activity times on a GPU, and there was neither gridlock resolution nor public transit) and was not applied

on large-scale networks (i.e., hundreds of thousands of links and nodes), it proved the concept of running an ABM on a GPU, as well as proposed what data structures to use on a GPU for this specific problem.

Perumalla et al. [77] developed a field-based GPU-accelerated mobility model for a large-scale road network with millions of nodes and links. The model converts a road graph into a grid of cells, where each cell propagates moving agents in either a horizontal or vertical direction with the pre-defined probabilities of turns. The model also supports queueing and dynamic re-routing as agents go into uncongested directions with higher probabilities. Results show that it takes 13 minutes to run a large-scale scenario of the state of Texas (USA), with 2 million nodes and 5 million links, and initially 10 vehicles per node. While the study demonstrates promising potential for traffic modelling on GPUs, it has some serious limitations. First, the behaviour of the agents is not modelled individually with pre-defined daily plans, but rather probabilistically. Second, the model is more applicable to evacuation scenarios because turning probabilities are defined based on the locations of evacuation exits. Third, the model loses granularity with the increase of the spatial scale due to limited texture memory of GPUs.

Shen et al. [78] implemented a microscopic traffic simulator with GPU acceleration to optimize traffic signal timing. However, the simulator was tested only on very small networks (four intersections) and there is no clear evidence of how activities and agent plans are organized, nor how feedback from the simulation could be fed into the agent's learning process. Lastly, the simulator was developed for a specific optimization problem and was not intended for large-scale scenarios.

Wang and Shen [79] presented a microscopic traffic simulation model running on a GPU. In contrast to many other works, the model implements a whole loop with the learning process between iterations included. Additionally, traffic lights can be modelled at intersections. The model was benchmarked only on small lattice networks, with sizes from 5 x 5 to 40 x 40 edges, and used randomly generated traffic. For the largest network, the achieved speed-up of the GPU-accelerated model was about 105 times greater than a CPU-based implementation. As there was no travel activity planning module implemented, each agent could only do a single trip. The model does not support multi-modal traffic, and it is not clear how it would perform using a large-scale real-world network.

Hirabayashi et al. [80] studied multiple design schemes of how to implement traffic simulations on GPUs. The main finding was that the more work is pushed to a GPU with less CPU-GPU synchronization, the more the runtime performance improves. While the authors implemented a limited, one-dimensional version of the optimal velocity model [81], the results show that a GPU can execute the mathematical background of the traffic model about 1 000 times faster than a CPU-based single-threaded version.

Sano and Fukuta [82, 83] presented a framework to assist the coding process of large-scale agent-based simulations with the focus on traffic. The framework provides a way to describe agents' behaviour and to automatically generate the code for execution on GPUs. While the framework is claimed to perform a light-

weight traffic simulation, no details have been reported. Instead, the works are focused on the planning activities of agents, specifically, path finding over a road network. Multiple routing algorithms have been implemented and benchmarked, and the results show that, while GPUs can handle routing in parallel, the runtime performance depends on hardware and requires tuning to be optimal. Later, Sano et al. [84] extended the work with a semi-automated performance tuning mechanism for the GPU-accelerated code. It was demonstrated that proper tuning can double the performance improvement for GPU-based routing.

Xu et al. [85] developed a GPU-based microsimulation framework called Entry Time-based Supply Framework (ETSF), and later the framework was improved [86]. The framework introduced a new method to reduce the time spent on processing vehicles during their movement across a network. The work showed a speed-up factor of 11.2 compared to a single-core CPU model on a small-scale artificial grid network (10 201 nodes and 20 200 links) with 100 000 vehicles. However, on a smaller network with the realistic topology of Singapore (3 179 nodes and 9 419 lanes) the proposed framework showed only a speed-up factor of 2.37 [87]. The evaluation of the published source codes [158] shows that ETSF functionality is limited, has scalability problems (almost all arrays have a predefined size, fixed length of routes, no feedback loop and convergence process, among others), and the published code was designed for a specific Singaporean scenario.

Vu and Tan [88] presented a GPU-based mesoscopic traffic simulation framework with improved performance compared to previous work [87] for Singapore network with 3 186 nodes and 9 437 lanes. Depending on simulated demand (100 000 to 300 000 vehicles), a speed-up factor of 5.8 to 6.5 compared to a sequential CPU-based version was achieved. Again, all demand is predefined and static without convergence (learning) process. The network cannot be considered large-scale, and there is no evidence that the framework can be developed beyond the prototype stage. Later, the authors extended [89] the framework with demand simulation on GPUs, although in a quite limited form with predefined sets of routes for OD pairs, basically implementing a logit model. Using the same Singapore area for benchmarks, the demand simulation achieved a five-times speed-up compared to a single-threaded CPU-based version. Other works related to demand simulation on GPUs [159] exist, but most of them implement only variations of a logit model.

Heywood et al. [90] recently attempted to implement mobility simulations with the FLAME GPU framework [91]. However, their prototype is very limited in scale and capabilities. Firstly, agents do not have daily plans, but instead are randomly moved on an artificial grid network. Secondly, the scale of the scenarios is very small – up to a 24 x 24 intersections network and up to 141 312 agents. Finally, agents do not learn during the simulation process. The focus of the work was to demonstrate the concept of a car-following model and visualization with FLAME GPU. Later, Heywood et el. [92] improved FLAME GPU with a general-purpose graph-based communication strategy to better adapt the framework for transport simulations. The experiments demonstrated that the improved framework achieved a speed-up factor of 43.8 compared to the multi-threaded commercial microscopic

simulator Aimsun [93] when running one hour traffic of 512 000 vehicles over an artificial square lattice network with a grid size of 512. Nevertheless, the limitations of the original framework remained.

Yedavalli et al. [94] developed a GPU-accelerated microscopic traffic simulator called MANTA. The simulator is capable of running metropolitan-scale simulations with a large network and millions of trips under 10 minutes of wall-clock time. The performance of the model has been evaluated using a San Francisco Bay Area (USA) scenario with a network of approximately 225 000 nodes and 550 000 links, and demand comprising about 3.2 million trips. The scenario was run from 05:00 until 12:00 with MANTA and the MATSim-derived simulator BEAM [160] with the discrete-event model JDEQSim from MATSim: the former was about 43% faster, accomplishing the task in about 4.6 minutes. While the performance improvement is not as significant as one might expect from a GPU-accelerated model, the authors note that the achieved granularity is greater than that of the mesoscopic model JDEQSim. However, MANTA has some serious limitations compared to BEAM and MATSim simulators. First, MANTA does not have a feedback loop with the learning process to converge a scenario to an equilibrium, and agents are not re-routed based on traffic jams; instead, agents use shortest routes based on distance. Second, MANTA simulates only car traffic, without public transit and other emerging modes like coordinated fleets. Third, only a static demand model is used where each agent has a single trip assigned, meaning that no activities are modelled and agents do not have time-dependent daily schedules. This approach also complicates implementing dynamic transport modes like taxis. Mitigating the above-mentioned limitations will likely increase MANTA's runtime as the performance of GPU-accelerated code is sensitive to changes of data structures and transfers of additional data to or from global memory or the host.

A more comprehensive survey on the application of hardware accelerators in agent-based models is available elsewhere [95].

2.3.2 *Queuing model*

One of the typical approaches to modelling traffic propagation through a network is a queue. There are various types of queueing models [161] in use. One is a point queue model [137, 162, 163] where vehicles are organized as a vertical queue by being stacked in the FIFO (first in, first out) queue of a link. The main drawback of this simple model is its non-realistic behaviour, including the lack of a spillover effect for congestion: a link has an infinite capacity, zero spatial length, and no inter-link interaction. In contrast, a spatial queue model [164–166] assigns the physical length and the corresponding storage capacity to a link, and therefore a spillover effect occurs as in the real urban road networks.

GEMSim utilizes a spatial queue model based on [167] and [54] that was adapted for massively parallel processing on a GPU. Although a GPU prototype of this queueing model was presented in [75], the current work extends this prior work

by making the queueing model practical for complex and large-scale scenarios: specifically, the current implementation improves the GPU data structures, a novel method to substantially increase the performance of inter-link communications, and provide built-in gridlock resolution and multi-modal support with a detailed public transit and coordinated fleets models.

The principle of the queueing model is presented in Figure 2.5. A link in the network is represented by two buffers: one buffer corresponds to the physical length of the link (spatial buffer), and the other buffer corresponds to the designed traffic flow (capacity buffer). A spatial buffer limits the number of vehicles that can simultaneously queue on a road segment, while a capacity buffer limits the number of vehicles that can leave the link within a time period.

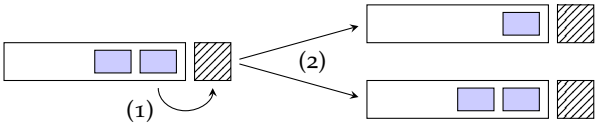


FIGURE 2.5: Queueing model: (1) moves a vehicle from the spatial buffer to the capacity buffer; (2) moves the vehicle from the upstream link to the downstream one.

A vehicle moves between the links in a two-step procedure. First, the vehicle moves from the spatial buffer to the capacity buffer if the following criteria are met:

- A vehicle has stayed for the minimum required time on the link. This minimum time is defined as $t_{link} = L_{link}/v_{link}$, where L_{link} is the physical length of the link, and v_{link} is the free-flow speed.
- A vehicle is at the front of the queue.
- Enough flow capacity has been accumulated: during the simulation step with a duration of t_{cycle} a link accumulates $q \cdot t_{cycle}/t_{period}$ of the flow capacity to keep the physical constraints, where q is a link capacity per time period t_{period}.

Second, when a vehicle is in the capacity buffer, the vehicle moves into the spatial buffer of the downstream link if there is enough free space. When a downstream link is connected to multiple upstream links, the order in which the upstream links are processed is defined to be uniformly at random and proportional to the flow capacity of each link:

$$p_k = \frac{q_k}{\sum_{i=1}^{n} q_i} \tag{2.15}$$

where q_i and q_k are the flow capacities of the i-th and k-th upstream links, respectively, and p_k is the probability of the k-th upstream link being selected. The selection of a downstream link is based on the route followed in the agent's daily plan.

As a mesoscopic model was used, there is no intra-link interaction between vehicles, but a spillover effect with traffic congestion occurs in accordance with the

queueing principles described above. The model still lacks some physical principles of traffic flow like kinematic wave propagation, but overall the model gives a good approximation for traffic flows in congested urban areas, as has been demonstrated in numerous studies [76].

The reason for using this model on a GPU is that each of the two propagation steps can be performed in parallel, and thus the model is well suited for GPU architecture. Indeed, in the first step, there is no data dependency between the links, while in the second step, there is no data dependency between the nodes: a capacity buffer of an upstream link is connected to the spatial buffer of a downstream link only through the same node.

2.3.3 *GPU simulation loop*

GEMSim has a time-step-driven GPU simulation loop with monotonically increasing simulation time t_{sim}, as presented in Figure 2.6. First, two kernels (a GPU function that is executed for all threads but with a different index for input data), `ProcessLinks()` and `ProcessNodes()`, perform network propagation by moving agents across the network $\mathcal{G}$ from input supply $\mathcal{S}$, while the kernel `ScheduleDemand()` schedules the agents for future execution. After reaching the simulation's end time t_{end}, the kernel `ScorePlans()` calculates the scores of plans, and externalities are collected by the `CollectExternalities()` kernel.

Input: $\mathcal{S}$, $\mathcal{D}_t$, t_{end}
Output: $\mathcal{S}_t^E$,$\mathcal{U}_t$

```
1  begin
2      for t_sim ⟵ 0 to t_end do
3          ProcessLinks(G, D_t, t_sim);
4          ProcessNodes(S, D_t, t_sim);
5          ScheduleDemand(S, D_t, t_sim);
6          UpdateExternalities(S, D_t, t_sim);
7          t_sim ⟵ t_sim + 1;
8      end
9      U_t ⟵ ScorePlans(D_t);
10     S_t^E ⟵ CollectExternalities(S);
11     return S_t^E, U_t;
12 end
```

FIGURE 2.6: Algorithm for the GPU-accelerated simulation loop of GEMSim.

The `UpdateExternalities()` kernel performs a periodic update of collected externalities when it is required. For example, some aggregated externalities like per-mode trip statistics are collected until the end of simulation, but some external-

ities like congestion are collected in recurring time periods and transferred back to host during the simulation. The reason for such incremental transfers is the occupied space in GPU memory. Hence, to avoid unnecessary memory waste on GPUs, some externalities are updated more or less frequently in the simulation loop, while others are transferred at once at the end of the simulation.

2.3.4 *Network propagation*

The kernel `ProcessLinks()` performs the first step of the two-step traffic propagation model, following the algorithm presented in Figure 2.7. Each GPU thread processes its own link denoted by index i. The function `GetScheduledTime()` retrieves the time $t_{enter} + t_{link}$ of an agent, where t_{enter} is the time when the agent has entered the last link.

```
   Input: G, D_t, i, t_sim
1  begin
2      AccumulateCapacity(G, i);
3      in_queue ⟵ GetSpatialBuffer(G, i);
4      out_queue ⟵ GetCapacityBuffer(G, i);
5      while (in_queue not empty) and (out_queue not full) do
6          if not HaveCapacity(G, i) then
7              return
8          end
9          agent ⟵ GetFront(in_queue);
10         if GetScheduledTime(D_t, agent) > t_sim then
11             return
12         end
13         RemoveFront(in_queue);
14         PushBack(out_queue, agent);
15         UpdateCapacity(G, i);
16     end
17 end
```

FIGURE 2.7: Algorithm for the `ProcessLinks()` GPU kernel to move vehicles from spatial buffers to capacity buffers.

The kernel `ProcessNodes()` performs the second step of the traffic propagation model, shown in Figure 2.8. Each GPU thread processes its own node denoted by index n. A special function `MultiModeNodeHandler()` delegates the flow control to the handlers of other transport modes like public transit and is discussed in detail in Chapter 3. The main purpose of the multi-modal handler is to give

other modes the possibility to implement a more complex logic when an agent shall be removed from the traffic propagation procedure (for example, parking activity). When an agent finishes the route and arrives at the place of activity, the function `ScheduleActivity()` schedules for how long the agent must perform the activity before continuing the execution of the daily plan. During the time of the activity, an agent is removed from the network propagation model. The function `ScheduleLink()` calculates time $t_{sim} + t_{link}$ when the agent can leave the spatial buffer of e_{next}.

A shortcoming of the original queueing model was the absence of gridlock resolution. A gridlock situation is typical for queue-based models when the head of the traffic flow hits the tail, and the whole flow is stuck. This also happens in real life when a road network has a grid structure (for example, the Manhattan area of New York, New York, USA) and some cars block intersections. This situation is resolved in the `ResolveGridlock()` function that checks when was the last time a car left a certain link. When this time exceeds a threshold, an agent is forced to try to queue in the link next to a blocked one in the route. While a gridlock is resolved, GPU threads are no longer independent and the threads need to synchronize an update of the downstream buffers with `PushBackAtomic()`. A gridlock resolution is essential for queueing models, otherwise agents can get stuck in very long and unrealistic queues for the rest of simulation time. In the gridlock resolution, a realistic behaviour of gridlocked drivers, whereby a driver finds a way to squeeze into the street, is implemented. For the sake of simplicity, only one of the gridlock resolution strategies is demonstrated here; other strategies will be discussed in detail in Section 2.5.

As mentioned before, the performance of the code executed on a GPU heavily depends on the data structures used in the GPU memory. The structure of the network $\mathcal{G}$ on a GPU is presented in Listing 2.4. The SoA approach is used to store per-link and per-node data. The size of global link buffers needs to be calculated in advance to pre-allocate memory on a GPU and to avoid poor performance and the limitations of GPU dynamic memory allocation.

The size N_l of a spatial buffer is calculated as follows:

$$N_l = max\{\left[k_l \cdot N_{lanes} \cdot \frac{L_{link}}{L_{veh}}\right], 1\} \tag{2.16}$$

where k_l is a spatial scaling coefficient, L_{veh} is the average length of the space occupied by a vehicle (a gross space of 7.5 meters is used for all vehicles), N_{lanes} is the number of lanes on the link, and $[\,]$ means integer. The size N_f of a capacity buffer is calculated as follows:

$$N_f = max\{\left[k_f \cdot q \cdot \frac{t_{cycle}}{t_{period}}\right], 1\} \tag{2.17}$$

where k_f is a capacity scaling coefficient. Both coefficients k_l and k_f are typically in the range $(0; 1]$, and are used for calibration purposes and for scenario downscaling.

```
   Input: S, D_t, n, t_sim
1  begin
2      E_in ⟵ GetUpstreamLinks(G, n);
3      for e_in ∈ E_in do
4          out_queue ⟵ GetCapacityBuffer(G, e_in);
5          while out_queue not empty do
6              agent ⟵ GetFront(out_queue);
7              if MultiModeNodeHandler(S, D_t, e_in, agent) then
8                  RemoveFront(out_queue);
9                  continue
10             end
11             e_next ⟵ GetNextLink(D_t, agent);
12             if e_next = ∅ then
13                 RemoveFront(out_queue);
14                 ScheduleActivity(D_t, agent);
15                 continue
16             end
17             in_queue ⟵ GetSpatialBuffer(G, e_next);
18             if in_queue is full then
19                 ResolveGridlock(D_t, agent);
20                 break
21             end
22             else
23                 RemoveFront(out_queue);
24                 PushBackAtomic(in_queue, agent);
25                 ScheduleLink(D_t, agent, e_next);
26             end
27         end
28     end
29 end
```

FIGURE 2.8: Algorithm for the `ProcessNodes()` GPU kernel to move vehicles from upstream links to downstream.

Link queues are organized as ring buffers. First, two global buffers are allocated for all links based on the pre-calculated N_l and N_f: one for spatial buffers (field `in_queue`), and the second for capacity buffers (field `out_queue`). Then, the global buffers are sliced for each of the links, and the partition information is stored in buffer descriptors. A buffer descriptor is a 64-bit integer (fields `in_descr` and

```
struct GpuNetwork {
    uint32_t in_queue   []; // Spatial buffers (global)
    uint32_t out_queue  []; // Capacity buffers (global)
    // Per-link
    uint64_t in_descr   []; // Descriptors of spatial buffers
    uint32_t in_state   []; // States of spatial buffers
    uint64_t out_descr  []; // Descriptors of capacity buffers
    uint32_t out_state  []; // States of capacity buffers
    uint32_t gridlk     []; // Time to track gridlock
    float    acc_flow   []; // Accumulated capacity
    // Per-node
    uint32_t link_count[]; // Number of upstream links
    uint32_t link_idx   []; // Indices of upstream links
    uint32_t link_st    []; // Start index of upstream links
};
```

LISTING 2.4: Structure of network graph $\mathcal{G}$ on a GPU with per-link queues and per-node upstream links.

`out_descr`) in which the high 32 bits define the offset q_{off} of the first element in the global buffer, and the low 32 bits define the size of the link buffer q_{size}. The dynamic state (or occupancy) of a link buffer is represented with a 32-bit integer (fields `in_state` and `out_state`) in which the high 16 bits define the offset q_{cur} relative to q_{off} of the first occupied slot in the buffer, and the low 16 bits define the number q_{cnt} of used slots in the buffer. Each slot in a buffer stores the index of the agent that is currently occupying the slot. To get a slot q_{next} for a new agent in a buffer:

$$q_{next} = q_{off} + ((q_{cur} + q_{cnt}) \mod q_{size}) \tag{2.18}$$

To remove an agent from the front of the buffer:

$$q_{cur} = (q_{cur} + 1) \mod q_{size} \tag{2.19}$$

A schematic of the memory layout for the link buffers is presented in Figure 2.9.

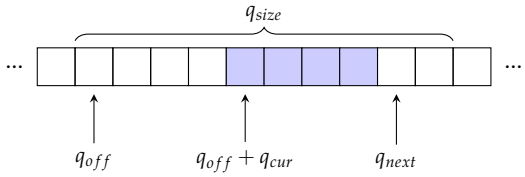


FIGURE 2.9: Layout of a link buffer in GPU memory partitioned in the global array with the descriptor variables of q_{off} and q_{size}, and the state of the buffer defined by the variables q_{cur} and q_{next}.

Other fields of the network structure are the following: `gridlk` is used to track the above-mentioned gridlock situations, `acc_flow` is used to accumulate per-link

flow capacities at each simulated time step, and the fields `link_count`, `link_idx` and `link_st` are used to describe upstream links for each of the nodes.

The idea behind allocation and partition of the global buffers is to reduce fragmentation: most of the GPUs allocate memory evenly to 256 bytes, and for large-scale networks with millions of links this leads to huge memory waste. The use of a ring buffer compared to a regular (sequential) buffer prevents expensive memory copy operations when the occupancy of the buffer changes and data movement is required, as instead, here only the state of the buffer needs to be updated. Both the buffer descriptor and the state have two smaller-sized variables combined into a single larger variable to guarantee that the buffer properties (either descriptor or state) can always be obtained with a single memory transaction, instead of issuing separate transactions for q_{off} and q_{size}, or for q_{cur} and q_{cnt}. Even though the amount of transferred memory is the same, the fewer number of transactions puts less pressure on the instruction pipeline of an SM.

While the `ProcessLinks()` kernel can achieve coalesced memory access to the link buffers, that is not the case for the `ProcessNodes()` kernel. To solve this issue, the following method of organizing access to the per-node upstream links was used. Let $\overline{E}_i \in E$ be an ordered set of upstream links (specifically speaking, indices of the links to access per-link buffers) of the i-th node, $i \in [1, \mid V \mid]$, and let $M = \max_i \mid \overline{E}_i \mid$. Also, let $\overline{E}^{(k)} = \{\overline{E}_1^{(k)}, \overline{E}_2^{(k)}, \ldots, \overline{E}_{|V|}^{(k)}\}$ be the ordered set of the k-th upstream links, $k \in [1, M]$, where

$$\overline{E}_i^{(k)} = \begin{cases} \overline{E}_i^{(k)}, & \text{if } k \leq \mid \overline{E}_i \mid \\ E_\emptyset, & \text{otherwise} \end{cases} \tag{2.20}$$

and $E_\emptyset$ is used as a `NULL` link item to ensure that $\mid \overline{E}^{(k)} \mid = \mid V \mid$ for a given network $\mathcal{G}$. The construction of sets $\overline{E}^{(k)}$ is performed after a two-way sorting process: first, nodes are sorted so that $\mid \overline{E}_i \mid \leq \mid \overline{E}_{i+1} \mid$, and second, the upstream links $\overline{E}_i$ of each node are sorted so that $p_k \geq p_{k+1}$, where p_k is from the Equation 2.15. Then, the ordered set $\{\overline{E}^{(1)}, \overline{E}^{(2)}, \ldots, \overline{E}^{(M)}\}$ defines an array of upstream links in GPU memory used by the `ProcessNodes()` kernel. This method maximizes the probability that two nearby GPU threads in a warp will perform a coalesced memory access when processing the upstream links, benefiting from a specific topology of typically sparse road graphs: most of the nodes have one or two upstream links, and only a few have more. The drawback is the redundant memory space $\sum_{i=1}^{|V|} (M - \mid \overline{E}_i \mid)$ used for $E_\emptyset$ when $\mid \overline{E}_i \mid < M$. The redundancy is introduced to keep a constant offset of size $\mid V \mid$ between two adjacent upstream links of the same node, which eliminates the need for an additional partitioning of the array.

2.3.5 *Demand scheduling*

The `ScheduleDemand()` kernel dispatches the agents into the network propagation model according to their plans. Each GPU thread processes its own agent denoted by the *agent* index. The demand scheduling algorithm is presented in Figure 2.10. Agents are modelled as finite state machines where each agent has a state and a role. In general, a state can be one of the following (other transport modes can extend the set of states):

- **Activity**, when an agent does not travel, but performs an activity.
- **Travel**, when an agent travels through the network.
- **Finished**, when an agent has finished their daily plan and has no travel legs left.

The role defines the type of an agent and can be one of the following:

- **Normal**, a synthetic agent from input data with the given daily-activity plan.
- **Transit**, an artificial agent created by the simulator and used as a driver of public transit vehicles.
- **DRT**, an artificial agent created by the simulator and used as a driver for vehicles in demand-responsive transport (DRT), i.e., coordinated fleets.

The role is needed to implement a special behaviour of the agents while the vehicles are propagated through the network. For example, a driver of a public transit vehicle has to check for stop locations and halt the vehicle to let people board or leave it; or a taxi driver has to wait for a person who has requested a ride. This different specific behaviour is implemented in a set of multi-modal handlers called from multiple GPU kernels.

To reduce the number of performed memory transactions, an agent has a scheduled time t_{sch}. It is guaranteed with this approach that until t_{sch} an agent keeps the state, therefore the agents with $t_{sch} > t_{sim}$ can skip the scheduling procedure. All scheduling functions in GEMSim set up t_{sch} to the corresponding value. An agent with states `Travel` and `Finished` can also skip the scheduling process. Additionally, by depending on transport modes other than a private car used by an agent, the `MultiModeDemandHandler()` can also force the demand scheduling procedure to skip the agent. A two-way sorting is applied to the population of agents to increase the probability that GPU threads in a warp proceed with the same code flow: first, by agent's role, and then by the expected arrival time. Sorting by a role allows GPU threads within a warp to execute a similar logic, while sorting by expected arrival times helps to balance the workload between the threads of the warp. When the agents are not sorted by the expected arrival time, the `ScheduleDemand()` has many warps where only a few threads are working and the rest are inactive because agents have already finished their daily-activity plans. Upon the departure of an

```
   Input: 𝒮, 𝒟_t, agent, t_sim
1  begin
2      if MultiModeDemandHandler(𝒮, 𝒟_t, agent, t_sim) then return;
3      if GetScheduledTime(𝒟_t, agent) > t_sim then return;
4      l_type ⟵ GetLegType(𝒟_t, agent);
5      if l_type = Network then
6          e_next ⟵ GetNextLink(𝒟_t, agent);
7          in_queue ⟵ GetSpatialBuffer(𝒢, e_next);
8          if in_queue not full then
9              PushBackAtomic(in_queue, agent);
10             ScheduleLink(𝒟_t, agent, e_next);
11             MultiModeDepartHandler(𝒟_t, agent);
12         end
13     end
14     else
15         MultiModeLegHandler(𝒮, 𝒟_t, agent, l_type, t_sim);
16     end
17 end
```

FIGURE 2.10: Algorithm for the `ScheduleDemand()` GPU kernel to dispatch agents into the network propagation according to their daily-activity plans.

agent, another multi-modal handler, `MultiModeDepartHandler()`, is called to allow for mode-specific behaviour. Finally, for non-network travel legs, the departure logic is handled in the `MultiModeLegHandler()` function.

The main structure of the demand D_t on a GPU is presented in Listing 2.5. The demand structure stores pointers to the current travel legs of the agents (field `cur`), and to the first travel legs (field `first`) in order to have the possibility of going through the travelled legs in the plan scoring procedure. Memory for travel legs is allocated in a single chunk and partitioned. The fields `sched`, `state` and `role` store per-agent scheduled time, current state and role, respectively. The local index (field `lidx`) is used for hierarchical indexing of multi-sampled demand, and will be explained later in the chapter. The last known velocities of the vehicles used by the agents are stored in the `velc` array, and the velocity tracking model is described in detail in Section 2.6. Therefore, the SoA approach is used here as GPU threads would require this information at each simulated step to decide which logic to execute next.

The `GpuBaseLeg` structure, presented in Listing 2.6, is used as a parent structure and is nested into other specialized travel leg structures of each transport mode. This nesting approach allows any specialized travel leg structure to be referred in the code by a pointer to the base structure; hence, other structures and the code are

```
struct GpuDemand {
    // Per-agent data
    GpuBaseLeg * cur  []; // Current travel leg
    GpuBaseLeg * first[]; // First travel leg
    uint32_t     sched[]; // Scheduled time
    uint32_t     lidx []; // Local index
    uint16_t     state[]; // State
    uint16_t     role []; // Role
    float        velc []; // Last known velocity

};
```

LISTING 2.5: Structure of demand $\mathcal{D}_t$ on a GPU with per-agent daily-activity plans and their personal state attributes used for scheduling and behaviour modelling.

more generalized. In the code, one can distinguish the exact type of the travel leg structure by using the `leg_type` field.

```
struct GpuBaseLeg {
    GpuBaseLeg * leg_next;  // Next travel leg
    int32_t      leg_type;  // Travel leg type
    int32_t      leg_start; // Actual start time
    int32_t      leg_end;   // Actual end time
    int32_t      veh_idx;   // Vehicle

    // Following up activity
    int32_t      act_start; // Start time
    int32_t      act_end;   // End time
    int32_t      act_dur;   // Duration

    int32_t      udata;     // User data

    // Variable length data...
};
```

LISTING 2.6: Base structure of a travel leg L_k on a GPU used to store the daily-activity plans of the agents.

Both SoA and AoS approaches were used, and the travel leg structure is organized as a linked list because the leg data is variable and depends on a transport mode. Each travel leg has a pointer `leg_next` to the next leg (or `NULL` for the last one), but the data of the next activity is embedded directly (AoS approach) into the leg structure using the fields `act_start`, `act_next` and `act_dur`. This is done to improve performance, as in most cases all fields of the activity structure (start time, end time, duration) are required for time scheduling. Memory transactions are always performed in chunks (the minimum is 32 bytes), and, typically, threads in a warp randomly access activity data; therefore using SoA will lead to three

scattered memory transactions instead of one, to load activity data for one thread. The recorded actual start and end times of a travel leg in the fields `leg_start` and `leg_end`, respectively, are also stored in the leg structure to be used later in the scoring procedure. Each travel leg can also have a vehicle linked to it through the field `veh_idx`, which is used to perform the leg; therefore, an agent can use multiple vehicles during a day. A travel leg can also carry some user-specific information in the `udata` field to allow a flexible customization of the travel modes.

A specialized travel leg structure, which is used for the general network-based mode, in addition to the base leg data, contains an array of link indices to traverse over the network in order to perform a route. In addition, the number of links and the currently traversing link index are stored. The size of the structure varies depending on the assigned route, and this is accounted for in the memory partitioning function.

2.3.6 *Network variations*

Network variations allow the definition of spatio-temporal changes in the input network during the simulation process. With the ongoing spread of intelligent traffic control systems and expected increase in penetration of AVs, there is a special need in the modelling of dynamic infrastructure. For example, lanes can be reversed in peak hours [168, 169] to increase flow capacity in a certain direction, or variable speed limits can be employed to improve traffic safety [170, 171]. Network variations can be supplied in a separate input file together with the scenario configuration.

Each variation describes one spatio-temporal change in the network, and specifies which network link is affected, in what way, and at what time. For now, the free-flow speed and flow capacity of a link can be changed through the variations. The implementation of the network variations is relatively straightforward. When variations are enabled, the host part of the GEMSim's simulation loop runs another, internal loop, that stores time-sorted variations and injects the required changes into the network when time comes. This, in general, generates low volumes of host-device data transfers as only a single value (four bytes) per variation has to be updated on a GPU.

2.3.7 *Iteration outputs*

At the end, the mobility simulator provides an input for the learning process $\mathcal{L}$. First, the `ScorePlans()` kernel is executed, where each GPU thread processes its own agent using temporal data that is recorded and stored in the travel leg structure. Afterwards, the kernel `CollectExternalities()` finishes aggregating externalities (traffic congestion, travel statistics by mode, occupancy of public transit routes, and others), and then this output is transferred from a GPU to the host.

Network congestion statistics are a mandatory output and are required for the learning process. However, for a large-scale network, and high temporal resolution, it may consume lot of GPU memory to collect the data. For example, assume that

a network $\mathcal{G}$ has one million links E, and congestion statistics (speed reduction, number of car passed) are collected in 15-minute periods. For a typical simulation of 30 hours, this will require the following amount of memory to be allocated:

$$M_{stats,net} = \left[\frac{t_{end}}{t_{stat,net}} \right]^{ceil} \cdot s_{cell,net} \cdot \mid E \mid \tag{2.21}$$

where $t_{stat,net}$ is the duration of a network statistical period, $s_{cell,net}$ is the size of data cell for one time period per link, and $[]^{ceil}$ means integer rounded up. Assuming only eight bytes is required per cell to store cell data (four bytes for speed reduction and four bytes for accumulated number of vehicles), then almost 1 GB of memory $M_{stats,net}$ is required. That is why network congestion statistics are collected in periods and transferred periodically to the host in the `UpdateExternalities()` kernel.

The structure used to collect network congestion statistics for a single time period has two arrays: one contains per-link aggregated travel time, and the second array holds the number of vehicles passed through. At the end of the simulation loop, the average per-link speed is obtained by calculating average travel time and using then the length of the link. The reason to not to store the speed directly is to reduce the number of computations done during the simulation, and the division operation is typically the slowest when compared to addition and multiplication.

Each time a vehicle passes a link, the time and counter fields are updated in the statistics by the corresponding GPU thread. While network links are processed in parallel by the `ProcessLinks()` kernel and the global memory is accessed in a coalesced manner pretty much, there is no guarantee that two links processed by neighbouring threads from a warp will have leaving vehicles at the same time. Therefore, the access to network statistics is mostly scattered. However, as the event of vehicle leaving a link does not happen each simulated time step for each link,the overall impact of statistics collection on the runtime performance is negligible.

Per-mode trip statistics are not a mandatory output, and, as they do not require extensive device memory, they are collected in a different way. By default, the duration of the time period for trip statistics is five minutes of simulated time. The data structure used to collect per-mode trip statistics is split into three parts: (i) temporal data aggregated for the current time period, (ii) the output data aggregated for the whole simulation, and (iii) some additional data required to manage two previous parts. The first part aggregates the number of departures, arrivals and en-route agents. The `UpdateExternalities()` kernel periodically copies this temporal data into the output part with aggregated values. After copying, the temporal arrays are reset to zeros, and the current time period is incremented. The output arrays store data successively per mode and then per time period; that is why some additional variables are required to calculate these offsets.

For example, for the m-th mode and i-th current period, the offset I_{trip} in the output arrays will be calculated as follows:

$$I_{trip} = m \cdot N_{periods} + i \tag{2.22}$$

where $N_{periods}$ is the total number of periods. As the amount of data to copy after each time period is relatively small, this update procedure for trip statistics is executed by a single GPU thread.

Compared to network congestion statistics, trip statistics consume much less memory:

$$M_{stats,trip} = \left\lceil \frac{t_{end}}{t_{stat,trip}} + 1 \right\rceil^{ceil} \cdot s_{cell,trip} \cdot N_{modes} \tag{2.23}$$

where $t_{stat,trip}$ is the duration of trip statistical period (by default, it is set to five minutes of simulated time), $s_{cell,trip}$ is the size of data cell for one time period per transport mode, and N_{modes} is the number of transport modes used in a simulation. Assuming that $N_{modes} = 5$ and $s_{cell,trip}$ consumes 12 bytes, $M_{stats,trip}$ will require only about 20 KB of memory. Similar to network congestion statistics, access by GPU threads to the structure is scattered. However, the events, which require update of trip statistics, happen even more rarely; only when an agent starts or finishes an activity. As one can see, the memory required for both congestion and trip statistics is known in advance and can be pre-allocated before the simulation starts.

There are other outputs generated by GEMSim, such as automatic comparison of simulated traffic with field counts provided through the input file. A file contains a list of stations with specified locations and the network links where traffic is being measured, and hourly traffic volumes from the fields.

2.4 PERFORMANCE ANALYSIS

As the increase in runtime performance of existing agent-based mobility simulations is one of the key motivations for this thesis, the developed framework was analysed for its performance from multiple perspectives using a large-scale scenario of Switzerland. While during the course of the thesis multiple daily-activity models for demand generation in Switzerland were developed, these benchmarks use an older version based on the Swiss Mobility and Transport Microcensus 2010 [172], and the methods to synthesize the trip chains are described elsewhere [173]. The population of agents consists of approximately 3.5 million agents with private cars, and approximately 1.7 million agents using public transit. The synthetic population of agents was derived from EnerPol's agent-based population model [131]. The distribution of the population across the country and the national road network are shown in Figure 2.11. The activity locations are based on Swiss Business and Enterprise Register 2013 [174] and Swiss Statistics on Enterprise Structure (STATENT) 2012 [175] and were selected for agents within the population model, and are fixed during the simulation.

The road network contains 513 770 nodes and 1 127 775 road links obtained from OSM. As the data quality in OSM varies, some required link attributes that are unreported are derived based on the road type at a link, or (in the case of flow capacities) derived from the speed limit and number of lanes. The public transit schedule of the year 2018, which contains 27 873 stops and 21 847 routes for trains,

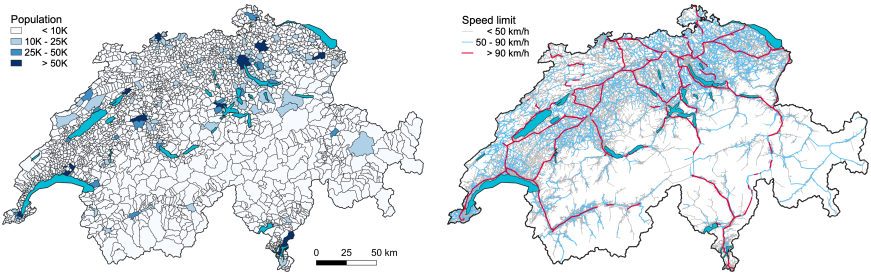


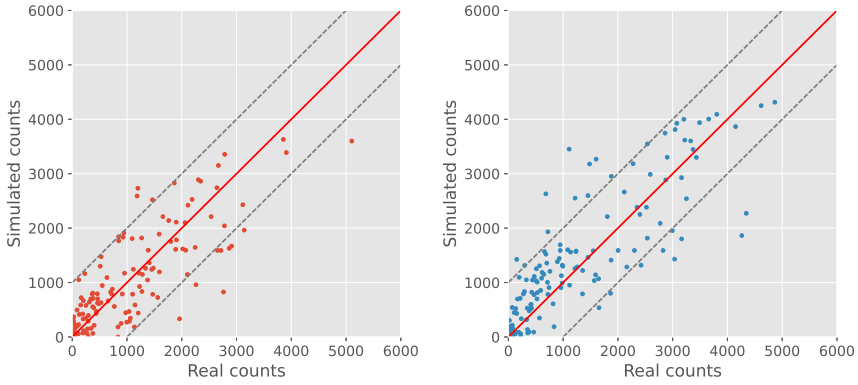
FIGURE 2.11: Distribution of the potential travel demand (left) and the road network (right) in Switzerland.

buses, tramways, metro and all other modes of public transit in Switzerland was taken from Swiss Federal Railways [176]. Public transit vehicles were not mixed on the links with private cars in this scenario. The calibration of the scenario was performed in the population model to match census data.

The scenario was validated against 137 traffic counts obtained from the Federal Roads Office of Switzerland [177]. The traffic count data includes directional hourly volumes for arterial highways and main roads for the full year of 2012. To converge to equilibrium, 100 iterations were performed with re-routing (based on congestion) 10% of agents between iterations. This number of iterations, whilst selected empirically, consistently gives good convergence. Comparisons of simulated and real counts for the morning (07:00–08:00) and evening (17:00–18:00) peak hours are presented in Figure 2.12: the main trend is captured.

Figure 2.13 compares the predicted and actual distributions of trip durations by car and public transit, respectively. The actual distributions of trip durations come from the Swiss microcensus data [172]. The agreement of the predictions to the data confirm that the large-scale scenarios are realistically modelled, as the simulations replicate the behavioural patterns of the agents. Therefore, the Switzerland scenario is used further to analyse the performance of GEMSim and to compare it with MATSim.

The performance of GEMSim was benchmarked with MATSim for multiple reasons. First, MATSim is a well-developed and highly optimized agent-based mobility simulator that exploits parallel computing. Second, MATSim's queue-based model QSim [76] is similar to that implemented in GEMSim, thus it is interesting to compare the performance of the developed GPU-optimized model with a CPU-optimized multi-threaded model, rather than with a CPU single-core implementation that is, as already explained, impractical in reality.



(a) Morning peak hour (07:00 – 08:00).

(b) Evening peak hour (17:00 – 18:00).

FIGURE 2.12: Comparison of simulated and real-world traffic counts in Switzerland using the older scenario.

2.4.1 *Considerations*

First of all, some considerations related to expected runtime performance of GEM-Sim must be taken into account. Computing power is required in only a few places of the GPU implementation, whereas the movement of agents along and between links take most of the time; therefore, the GPU implementation has some memory- and latency-bound limitations and the performance depends strongly on the memory bandwidth of the GPU device. One limitation is in the `ProcessNodes()` kernel, where threads in a warp access agents and downstream links in a non-coalesced manner. Another limitation is in the `ScheduleDemand()` kernel, where agents are injected into the simulation loop; while the data of each agent is coalesced, the queues of the links are not coalesced.

Notwithstanding the above, it is interesting to note that whereas the performance of the `ProcessNodes()` kernel depends on the size of the network, the performance of the `ScheduleDemand()` kernel depends on the size of the population sample.

As mentioned in the previous chapter, it is possible to route global memory requests by bypassing the L1 (on-chip) cache; this improves performance for non-uniform (that is, scattered) access patterns, and the separate compilation and linking mechanism from CUDA has been employed to tune the performance of the presented kernels. This mechanism allows compiling each kernel with different flags and link the code together afterwards. The following kernels were compiled with the L1 cache turned off:

- `ScheduleDemand()`
- `ScorePlans()`

And the following kernels were compiled with the L1 cache turned on:

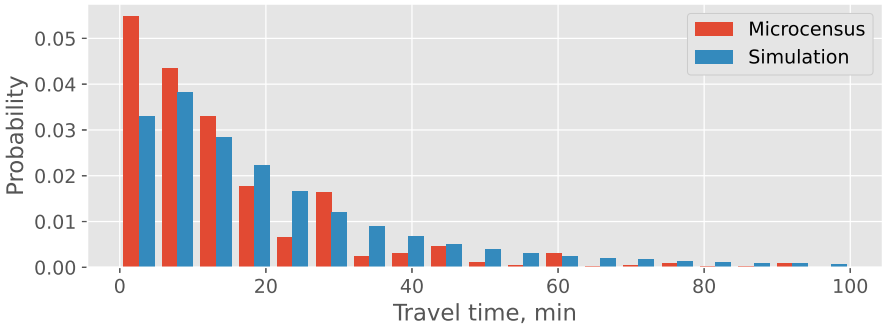


FIGURE 2.13: Distribution of travel time for a typical working day, microcensus and simulation of the older Switzerland scenario.

- `ProcessLinks()`
- `ProcessNodes()`
- `UpdateExternalities()`
- `CollectExternalities()`

Regarding the L1 cache, as none of the simulator's code uses shared memory capacity, hardware resources were only allocated to the L1 cache. Experiments showed that such tuning of the L1 cache gives up to 10% improvement in the runtime performance of the GPU-accelerated code, especially when GPU devices with full L1 cache line promotion are used.

2.4.2 *Convergence*

The scenario was evaluated in terms of the convergence to equilibrium, to demonstrate that the platform, and the learning part in particular, performs as expected. The convergence to equilibrium of a simulated scenario was evaluated at both aggregated and disaggregated levels. Figure 2.14 shows the average score of the agents for each iteration. While initially the score is as low as 15, it quickly increases asymptotically to 32.5 in 20 iterations. After 20 iterations, the score remains stable except for a few iterations around an iteration count of 50 when a subset of agents have poor re-routing. Notably, after 80 iterations when agents stop innovating their plans, the score increases slightly and remains stable until the end of the simulation. Figure 2.15 also shows the number of agents en-route during the day for different iterations. While at the first iteration most agents choose the same routes and are thus stuck in congestion, after 20 iterations the agents learn where are the congested areas and avoid these areas when re-routing; hence almost all agents can reach home by the end of the day. For example, in the area of Zurich city which has a

population of about 400 000 people, the average duration of a car trip is reduced from 28 to 20 minutes after the relaxation of road congestion, while the average trip distance is increased from 15 to 19 km.

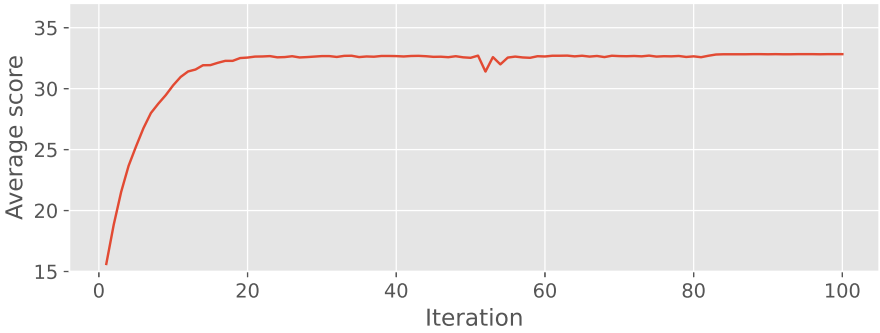


FIGURE 2.14: Average score of agents between iterations when converging the older Switzerland scenario.

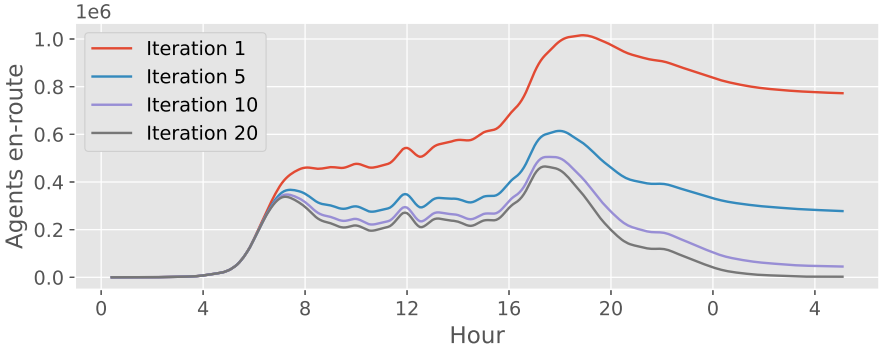
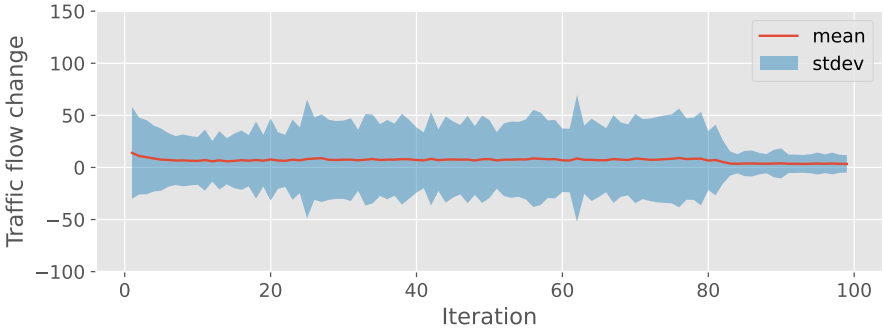


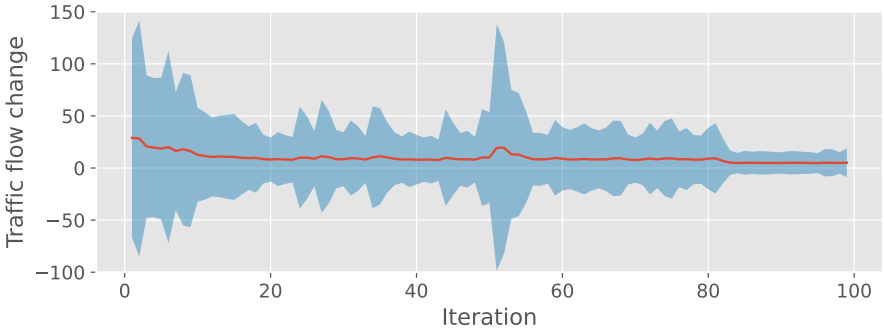
FIGURE 2.15: Agents en-route between iterations when converging the older Switzerland scenario.

The convergence at a disaggregated level was evaluated by comparing traffic flows during the morning (07:00–08:00) and evening (17:00–18:00) peak hours along links with positive traffic flows which are designated as motorways, and primary and secondary roads in OSM. A total of 32 441 links were compared for the morning peak hour, while for the evening peak hour 31 849 links were compared. At each iteration, except the first, the traffic flows were compared to the traffic flows from the previous iteration. Figure 2.16 shows the mean absolute difference and standard deviation for each of the iterations. In the morning both mean and standard deviation are initially high, but decrease in subsequent iterations. However, after about 10 iterations, the mean difference stabilizes, but the standard deviation increases. This is because,

after the initial improvement, agents perturb the system more by changing their daily plans. After 80 iterations, agents stop actively innovating daily plans and the standard deviation drops by a factor of four and subsequently remains stable. In the evening, the behaviour of the system is similar; however the magnitude of the standard deviation is higher than in the morning. Interestingly, there is a sharp spike in the standard deviation around 50 iterations that also corresponds to the decrease in the average score of the agents that was observed before in Figure 2.14.



(a) Morning peak hour (07:00–08:00).



(b) Evening peak hour (17:00–18:00).

FIGURE 2.16: Difference of traffic flows along links between iterations when converging the older Switzerland scenario.

2.4.3 *Simulation loop*

As the systematic design and implementation of a GPU-accelerated mobility model were among the main goals of the thesis, the runtime performance of the whole simulation loop was first evaluated and compared with MATSim using a sample of 3 million agents who are car drivers only. To run a scenario with both simulators, a cluster node with two AMD EPYC 7742 CPUs clocked at 2.25 GHz, 1 TB of RAM

and Nvidia A100 (40 GB DRAM on-board) GPU was used, running CentOS Linux 8.2 (x64 build). An iteration was run for one full day from 00:01 AM until 06:00 AM of the next day. MATSim with the QSim multi-threaded CPU-based queuing model (7 threads for the queueing model itself and 14 threads for events handling) was used for comparison of performance. The learning process performed re-routing for 10% of the agents using 20 threads in both simulators. GEMSim also used 20 threads to parallelize other tasks like data binding. Performance results for different parts of the simulation loop are summarized in Table 2.2. Compared to a similar benchmark performed with both simulators earlier [178], the following changes were made:

- More recent hardware and software were used, and include one of the top CPUs currently on the market, as well as a top available GPU. This ensures that both simulators can get enough computing power, while before lower-end CPUs have been used.

- Both simulators were run using the *numactl* utility that binds the CPU threads of a process to a single CPU socket and to the nearest (in terms of latency) memory banks of RAM. This ensures that both simulators allocate hardware resources optimally on the NUMA node.

- Both simulators now keep three last plans for each of the agents, instead of only two as before. While creating stronger pressure on consumed RAM, this is a more realistic setting – typically, three to five plans are used, and the default value for MATSim is five [76].

- More iterations (ten instead of five) were run before making the measurements. Again, this reflects a more realistic use case when dozens to hundreds of iterations are run, generating higher memory pressure and more relaxed traffic flows. In previous studies, only two plans per person and fewer number of iteration were possible due to the high RAM usage of MATSim and less RAM installed in the testing node.

It is interesting to note that GEMSim requires more time to bind all the input data to a GPU than to run the mobility simulation on a GPU itself, while, in contrast, MATSim requires much less time to prepare the input data for the simulation. It is thus evident that the design of a simulation loop with GPU acceleration as a whole is important to consider, and the implementation of a GPU-accelerated traffic propagation model itself does not necessarily mean an overall speeding-up of the same level. Therefore, while many other works in the area of GPU-accelerated mobility models focus mainly on the improvements of the runtime performance of the traffic propagation, the major bottlenecks of the whole loop are left outside of the scope. This leads to situations when speed-up factors, achieved and reported for prototype models, do not reflect the overall performance improvement of simulations in practice. An example can be seen in previous work of MATSim developers

TABLE 2.2. Comparison of runtime performance of GEMSim and MATSim for each part of the simulation loop.

Step in the loop	GEMSim, s	MATSim, s	GEMSim speed-up
Data preparation	100	21	0.21
Mobility simulation	90	9058	100.64
Scoring	1	4	4.00
GPU output data	47	–	–
Learning	213	1149	5.39
Total	451	10232	22.69

[76], where a performance gain of a GPU implementation was lost due to the cost of data transfers between a GPU-accelerated part of the code and the rest of the code.

GEMSim's mobility simulation runs 100 times faster than MATSim's QSim model (measurements are performed after the first 10 "warm-up" loop iterations), the more efficient implementation of re-routing has superior performance, and in total the whole simulation loop of GEMSim is more than 22 times faster than the simulation loop of MATSim.

Another interesting point to note is that MATSim runs slower compared to the previous benchmarks, despite the current hardware being more powerful. This can be explained by the fact that the more relaxed traffic flows become, the more time is required for MATSim to process generated events. The number of events increases as agents learn how to travel and do so better, hence fewer agents are stuck in traffic by the end of the day. Similarly, GEMSim has a runtime performance drop on later iterations, from about 60 to 90 seconds, as more memory transactions on GPUs are required to move agents across the network. However, as GEMSim is not an event-based simulator, the runtime performance stabilizes relatively quickly at around 90 seconds per iteration.

2.4.4 *Scalability*

First, the runtime performance of GEMSim simulations with different population samples was evaluated on a cluster node with two Intel Xeon E5-2620 v4 CPUs clocked at 2.60 GHz, 256 GB of DRAM and four Nvidia P100 (16 GB RAM on-board) GPUs. In total, 20 CPU threads were used to run a scenario. The runtime of one simulated daily iteration for each of the population samples is shown in Figure 2.17. Agents were sampled uniformly at random from the full-scale scenario, including car drivers and public transit users. The runtime performance scales non-linearly, meaning that when the size of a sample increases from 20% to 40%, the runtime increases from 220 to 350 seconds only. The cause of this behaviour is the structure

of a GPU-accelerated simulation loop. There are two main performance bottlenecks beyond traffic propagation: (i) data transfers between a GPU and the host side, and (ii) re-routing of the agents between iterations. Both bottlenecks almost linearly depend on the size of a simulated population, but the traffic propagation part, which is accelerated with a GPU, has a non-linear runtime dependency. Thus, the overall runtime of a simulated iteration has a non-linear dependency on the size of a population sample.

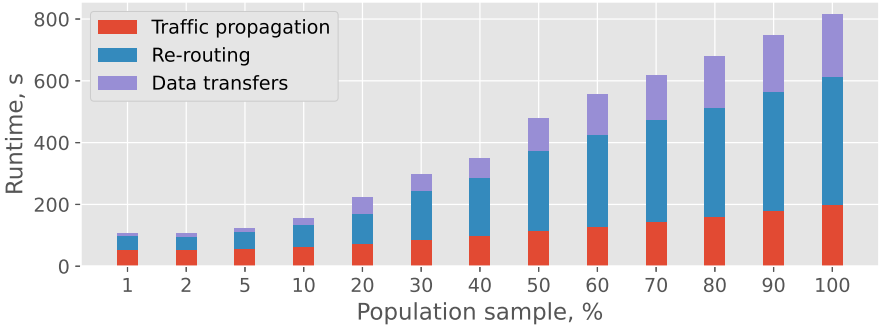


FIGURE 2.17: Runtime performance of GEMSim for one simulated iteration depending on population sample size.

Next, the scalability of both simulators, GEMSim and MATSim, was evaluated by using population samples from 100 000 to 3 million agents. For the traffic propagation part, Figure 2.18 shows that GEMSim yields a significant reduction in time for large-scale scenarios. It is evident that in comparison to MATSim, GEMSim has a real-time ratio two orders of magnitude higher (i.e., how many seconds of scenario time are simulated in each second of real time). While MATSim quickly drops to ratios of around 20 with a population size of 1.5 million, GEMSim still has ratios about two orders of magnitude higher or more even for larger population samples.

Figure 2.19 shows the speed-up of GEMSim over MATSim for the traffic propagation part, and for large population samples the speed-up factor is between 80 and 100. It is also interesting to observe that GEMSim has a performance saturation point around the 2 million population sample, probably because the number of agents becomes greater than the number of network links, and then the runtime of the GPU procedure that updates states of the agents prevails. Nevertheless, the speed-up factor continues growing with the further increase in population sample size.

The host RAM consumption, shown in Figure 2.20, is about five times lower for GEMSim for large population samples. Moreover, MATSim's RAM consumption increases much more rapidly as the population size increases.

Figure 2.21 shows the GPU DRAM consumption for the different population samples. One limitation of GPU-accelerated models is the relatively small amount

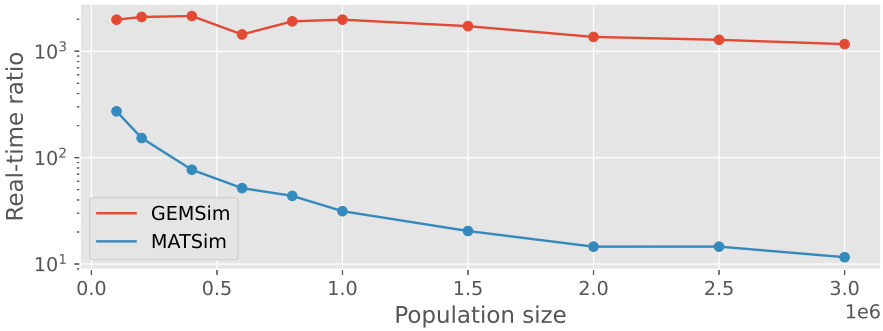


FIGURE 2.18: Real-time ratio (simulated seconds in each second of real time) of the traffic propagation part for GEMSim and MATSim.

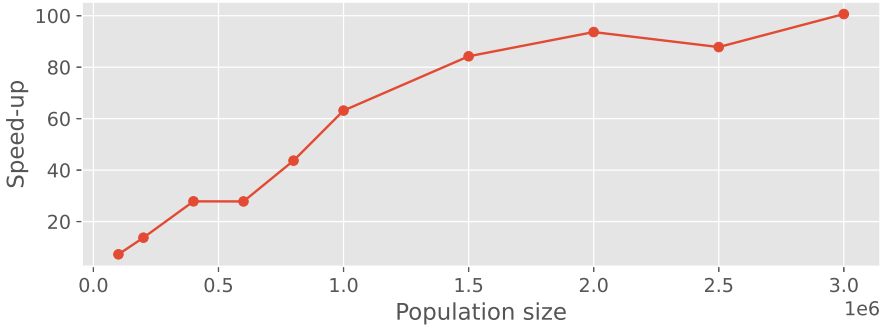


FIGURE 2.19: Speed-up factor of GEMSim over MATSim for the traffic propagation part depending on population sample size.

of available on-board memory (currently, up to 80 GB compared to TBs of RAM available for CPUs). The Switzerland scenario, with this travel demand model and car-driving agents, uses about 6.8 GB of GPU memory, which includes: 6.00 GB for the daily-activity plans of the agents, 0.80 GB for the multi-modal network (mostly for per-link queue buffers), and some relatively small amounts of memory for the public transit schedule, per-vehicle data and aggregated statistics. Therefore, the scalability of the GEMSim also depends on the available GPU memory, and the simulator can run up to 40–45 million agents on a single GPU using currently available hardware.

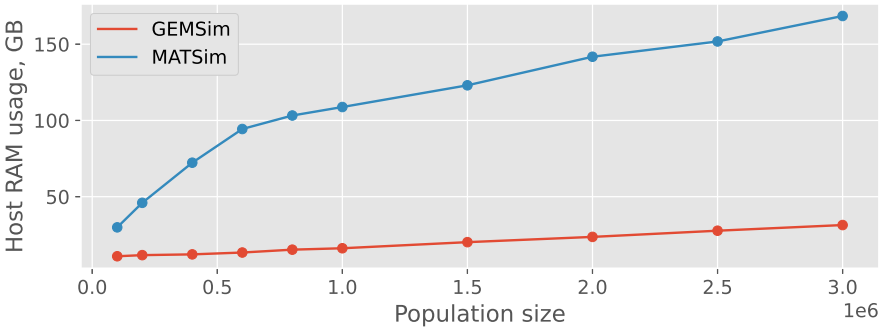


FIGURE 2.20: Peak host RAM consumption during the simulation for GEMSim and MATSim depending on population sample size.

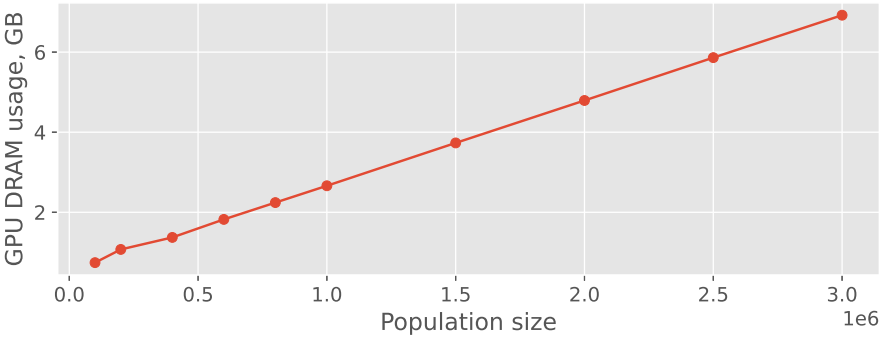


FIGURE 2.21: Peak GPU DRAM consumption by GEMSim during the simulation depending on population sample size.

2.4.5 *Kernel profiling*

CUDA allows the profiling of GPU-accelerated applications in order to evaluate bottlenecks and to fine-tune kernels for better performance. Profiling also allows us to understand how well selected data structures and algorithms fit the hardware, and what are the limitations applied. GEMSim was profiled using Nvidia Nsight Compute (version 2022.1.0) when simulating the morning peak hour of the older Switzerland scenario. Table 2.3 describes the main performance metrics collected, while Table 2.4 presents the actual measurements from the profiler.

TABLE 2.3. Description of main performance metrics collected during the profiling of GEMSim GPU kernels.

Metric	Description
Duration (microseconds)	How much time it takes to execute a kernel.
Compute Throughput (%)	Achieved utilization of compute resources. Low values may indicate underutilization.
Memory Throughput (%)	Achieved utilization of memory bandwidth. Low values may indicate underutilization.
L1 Cache Hit (%)	The rate of successful L1 cache hits for memory transactions. A low hit rate indicates scattered access patterns. Access to L2 cache imposes higher latencies.
L2 Cache Hit (%)	The rate of successful L2 cache hits for memory transactions. A low hit rate indicates scattered access patterns. Access to global memory in case of miss imposes highest latencies.
Occupancy / Theoretical (%)	The ratio of maximum number of active warps to maximum active number of warps supported by SM. Low occupancy may indicate workload imbalances of a kernel or scheduling overheads.
Branch Efficiency (%)	Proportion of branches when all threads in a warp select the same branch target. Low efficiency leads to serialization of instructions.
Warp Efficiency (no eligible, %)	Number of active cycles when no instructions have been issued. High values may indicate stall issues and underutilization of hardware resources.
Warp Cycles per Instruction	Average number of cycles between issuing two instructions for a warp. Large values may indicate scattered memory access and latency issues.
Long Scoreboard Stall (%)	How often a long scoreboard dependency was among the warp stall reasons. Long scoreboards are reported for instructions that may leave an SM (i.e., global or texture memory).
Registers [/65536]	Number of registers used out of the maximum available. Overuse of registers leads to reduced occupancy and spillover to global memory, causing stall issues.
IPC [/4.0]	Instructions per cycle (IPC) executed out of the peak possible.
Instruction Serialization (%)	Ratio of serialized instructions, due to latencies, memory or atomic operation conflicts.

Table 2.4. Profiling results of the main GEMSim GPU kernels used for traffic propagation and the scheduling of agents' daily-activity plans.

Metric Kernel	**ProcessLinks()**	**ProcessNodes()**	**ScheduleDemand()**
Duration (microseconds)	50.98	144.10	715.81
Compute Throughput (%)	20.32	13.85	5.01
Memory Throughput (%)	55.63	24.34	7.25
L1 Cache Hit (%)	32.67	61.82	49.01
L2 Cache Hit (%)	20.90	48.05	59.42
Occupancy / Theoretical (%)	67.15 / 100.00	35.81 / 50.00	33.91 / 50.00
Branch Efficiency (%)	72.31	82.53	99.32
Warp Efficiency (no eligible, %)	77.07	84.09	94.75
Warp Cycles per Instruction	47.30	36.93	103.78
Long Scoreboard Stall (%)	79.60	83.00	90.30
Registers [/65536]	44 032	47 104	58 528
IPC [/4.0]	0.89	0.60	0.21
Instruction Serialization (%)	1.41	3.18	3.51

Duration. It is clear that the kernel `ScheduleDemand()` is the main runtime bottleneck, with an average duration of about 715.81 microseconds, contributing about 78% of the total GPU execution time. The `ProcessLinks()` kernel has the shortest runtime of about 50.98 microseconds, which is what is expected as it has relatively simple logic combined with coalesced memory access. The total runtime is about 910 microseconds; however, this time reduces towards the end of a simulation because as more agents arrive back home, fewer have to be scheduled.

Compute throughput. As mentioned before, mesoscopic mobility simulators typically do not perform enough computations to saturate a GPU with work. That is the reason why, overall, only about 20% of compute throughput has been achieved by the `ProcessLinks()` kernel. However, most of these operations are coming from load-store units that have to generate addresses while loading and storing instructions. The lowest compute throughput of about 5% is achieved by the `ScheduleDemand()` kernel due to the few calculations required, as well as high memory latencies.

Memory throughput. The `ProcessLinks()` kernel achieved the highest memory throughput of about 55%, which means that this kernel is memory bound, while others are latency-bound. Again, the demand scheduling kernel has the poorest runtime performance due to scattered memory access patterns. The reason why the `ProcessLinks()` kernel is so efficient in memory access is that for most of the execution time it has a coalesced access with low latencies, and the logic of this kernel is the most straightforward with only a few conditions to be evaluated; it is also the smallest kernel in terms of the code base.

L1 and L2 cache hit. Interestingly, the `ProcessLinks()` kernel has the lowest hit rate of between 20% and 32%, while other kernels are in the range of 48% to 61%. The lowest hit rate happens because the kernel already has a coalesced access, so it only needs to fetch data from global memory in those few cases when scattered access happens. The `ProcessNodes()` kernel has the best hit rate because it often reads multiple consecutive fields from data structures with travel demand, so accessing one fields automatically caches some other fields. However, while it seems that the `ScheduleDemand()` kernel has efficient caching behaviour, this is misleading. First of all, as mentioned earlier, this kernel compiles with L2 cache bypass, and it does not cache access to global memory there. However, the demand scheduling procedure requires access to many variables at the same time as making extensive use of the registers (59 registers per a GPU thread); therefore, a spillover occurs moving some data back and forth to global memory. These transactions generate some cache hits, and not the kernel's code explicitly.

Occupancy. The `ProcessNodes()` kernel has relatively high occupancy of almost 70%, which means that neither the resources of SM, like registers or shared memory, nor the kernel launch configuration (block or grid size) are limiters of performance, but rather latencies of memory access. In contrast, two other kernels have a maximum theoretical occupancy of only 50% due to extensive use of registers. As each GPU thread uses too many registers, and it is a limited resource of SM, less warps can be active at SM, reducing the total achieved occupancy.

Branch efficiency. For all kernels, the branch efficiency is relatively high, especially in the demand scheduling where it goes above 99%. Such high branch efficiency for the demand scheduling can be explained by the fact that the moment when an agent actually requires a scheduling action is a relatively rare event, as on average agents have about three to four travel legs. Therefore, most of the warps only check the status of the agents and exit the kernel execution. On the other hand, network propagation kernels have more non-uniform decisions across the threads of a warp.

Warp efficiency. A warp is considered to be eligible if it can issue the next instruction. At each cycle a warp scheduler tries to select an eligible warp to issue an instruction: warps that are not eligible are considered to be stalled, for multiple reasons. All kernels have high share of non-eligible warps at each cycle, which indicates high latencies from stall issues. As the kernels are memory- and latency-bound, this is an expected behaviour.

Warp cycles per instruction. This metric indicates how long it takes for a warp to issue one instruction after another. Large values, like for the `ScheduleDemand()` kernel, typically indicate stall issues as a warp cannot proceed to the next instruction. Again, for a non-compute-bound kernel, this is expected.

Long scoreboard stall. While from the previous metrics it is clear that the kernels experience stall issues, this metric proves that the majority of the stall issues are due to waiting long scoreboard dependencies; that is, global memory access. This stall issue constitutes from 79% to 90% of all stall issues, depending on the kernel. Therefore, in order to optimize the kernels further, one has to improve memory access patterns, or reduce the amount of data to read and write.

Registers. As mentioned, only the `ProcessLinks()` kernel fits the size of a register file and is not limited in terms of maximum achievable occupancy, using 32 registers per thread. Kernels `ProcessNodes()` and `ScheduleDemand()` consume 64 and 59 registers, respectively. The main reason is that these kernels have more complex logic with branching and increased number of variables to evaluate.

IPC. For all kernels, the number of instructions per cycle is low, reaching down from 0.89 for the `ProcessLinks()` kernel to 0.21 for the `ScheduleDemand()` kernel, for which the primary reason is high memory latencies, incurring long waiting times. Essentially, IPC is inversely proportional to memory latencies in the current application. As the demand from kernels for arithmetic operations is low, there is nothing to execute for warps while waiting for memory transactions to finish. This is also supported by the fact the most frequent instruction for the kernels is integer multiply-and-add, which is actively used to generate memory addresses.

Instruction serialization. Typically, instruction serialization happens when a GPU wants to "replay" an instruction for some reason, most often long stall times or memory conflicts. It seems that such events happen rarely in traffic simulations, and the serialization varies in the range of 1% to 4%. This means that code branching does not cause intensive serialization rates, and shared memory is not used by the kernels. There are, however, atomic operations in gridlock resolution (`ProcessNodes()`

kernel) and demand scheduling parts, when agents compete for access to the same link buffers. This is why for these kernels the overall serialization is higher.

Overall, profiling confirms that the GPU kernels are either memory- or latency-bound, with little use of arithmetic resources. Therefore, to improve the runtime performance further, one needs to improve memory access by making it less scattered, or reduce the number of memory transactions, for example by compressing transferred data and using underutilized arithmetic resources for decompression. On the other hand, as only a small share of compute throughput is utilized, more complex and detailed modelling algorithms can be implemented to make models more realistic.

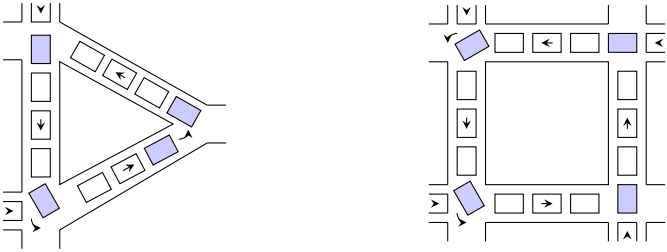
2.5 GRIDLOCK RESOLUTION

This section elaborates possible ways to effectively resolve artificial gridlocks in GPU-accelerated mobility simulations. This is accomplished by providing different gridlock resolution strategies that are specific to GPU-accelerated traffic queue models. Nevertheless, these strategies can also be applied to traffic models executed on CPUs in order to improve the overall performance by reducing the number of memory allocations.

2.5.1 *Background*

Traffic gridlock occurs when a vehicle at the front of a queue of traffic can neither move forward nor turn because the traffic is blocked (or the flow of traffic is very slow) in all directions for the local area. Traffic gridlock is not uncommon when roads are laid out in a grid pattern and cars block intersections (for example, Manhattan Island in New York, New York, USA), or at a roundabout junction of roads. A schematic of gridlock situations is shown in Figure 2.22. The physics of gridlocks and ways to mitigate them have been extensively studied [179–182]. However, in practice, there is a set of very simple and efficient measures that are applied to prevent gridlock, such as properly configured traffic lights with turn restrictions, or fines for blocking an intersection.

In road traffic simulations, the implementation of real-world rules can be challenging: these rules range from the complexity arising from human behaviour to computational limitations. Moreover, while in reality a gridlock typically happens in highly congested areas, in microscopic simulations artificial gridlocks may occur. These artificial gridlocks may lead to a completely jammed traffic flow with most of the agents getting stuck, putting the simulation into an abnormal and unrealistic state. Reiser and Nagel [183] used the MATSim [76] framework to study the phenomenon of network breakdown with a mesoscopic traffic queue model. They showed that an iterative learning process in which agents adapt their behaviour to the environment (that is, re-routing of a daily plan based on observed congestion) between simulated daily iterations leads to the oscillatory process of a network



(a) Triangular gridlocked section. (b) Rectangular gridlocked section.

FIGURE 2.22: Gridlock situations: arrows show the intended directions of traffic flows, and the filled vehicles completely block the traffic flows.

breakdown. This network breakdown, which was observed to occur at random, led to a global and unrealistic gridlock. Rickert and Nagel [184] observed gridlock situations while performing microscopic simulation of Dallas-Fort Worth, Texas (USA), using TRANSIMS [101]. The authors suggested reducing the green light phase of traffic lights by about 40%, which improved gridlocking behaviour while keeping the relative delays of the agents optimal. Nagel and Barrett [140] also showed that gridlocks in microsimulations can be mitigated by removing stuck vehicles from a simulation and performing iterative re-routing of the agents.

Horni and Axhausen [185] showed that the use of queue models in traffic simulations may lead to artificial gridlocks because of the predefined behaviour of the agents (that is, an inability to dynamically re-route when a gridlock occurs); thus, two strategies to resolve gridlocks were proposed:

- Increase the physical capacity of simulated streets (links) by up to a factor of five; or
- Push stuck vehicles into downstream links even if there is no available free physical space.

The second approach is implemented in the current version of the MATSim's queue model, QSim, through the dynamic extension of queues based on the spatial capacities of the network links. When one implements a traffic queue model that is executed by a CPU, the above-mentioned approaches can be easily implemented using dynamic memory allocation, even though the impact of such implementations on the simulation outcomes remains an open question. With GPU-accelerated mobility simulators, it is more challenging to resolve artificial gridlocks because dynamic on-device memory allocation is unavailable.

2.5.2 *Resolution strategies*

A vehicle moves along a route that comprises a sequence of links. A potential gridlock situation arises when a link accumulates enough flow in a capacity buffer, and therefore a vehicle cannot move to a downstream link because no free space is available for a simulated time duration longer than t_{stuck}. While configurable, currently, t_{stuck} is set at 10 seconds based on simulation experiments reported in the previous work [185]. The following strategies to resolve a gridlock were proposed and evaluated:

- **push-through**: a vehicle skips the congested link immediately in front of it and tries to enter into the next downstream link along its route. If the next link is full, then at the next simulation step the vehicle will try to progress even further downstream, and so on. This strategy was mentioned in Subsection 2.3.4, as one of the reference implementations.
- **re-inject**: a vehicle is removed from the capacity buffer and, starting from the next simulation step, tries to enter the next link in the demand scheduling kernel, thus competing with other vehicles for space in the queue in an atomic compare-and-swap operation.
- **squeeze**: each link has a pre-allocated virtual capacity of constant size in the spatial buffer, and a vehicle tries to use the virtual capacity of the downstream link. Virtual capacity can be used only by stuck vehicles.

It should be noted that these strategies were designed for GPU execution without dynamic on-device memory allocation. The *push-through* strategy, although very simple to implement, is the least realistic strategy, and furthermore, this strategy makes visualization of simulation outcomes difficult. The main disadvantage of the *push-through* strategy is that the `ProcessNodes()` kernel can no longer be executed completely in parallel, and GPU threads have to use atomic operations to reserve space in downstream queues. The *re-inject* strategy is a more realistic strategy as vehicles do not make spatial leaps, but visualization of simulation outcomes remains a problem. Finally, the *squeeze* strategy is the most realistic of the three strategies, and allows for easy visualization of simulation outcomes. However, this strategy requires that additional memory on a GPU is pre-allocated for the virtual capacity. A constant virtual capacity (that is, defined in the configuration of a scenario) was used for all links in order to avoid extra memory transactions and calculations when GPU threads work with link buffers. One can also note that the two former strategies make the simulations more stochastic because the sequence of vehicles along the same route departing at the same time is not always the same.

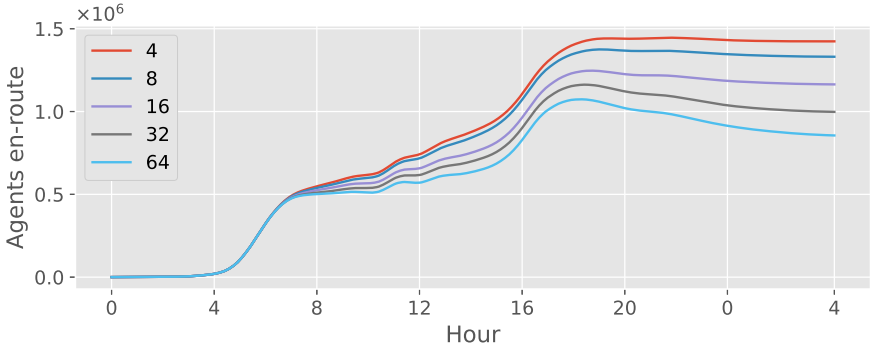
2.5.3 *Impact on simulations*

To evaluate the three gridlock resolution strategies, a large-scale scenario for the whole of Switzerland, described in Section 2.4, was used. Here, while 3.1 million cars

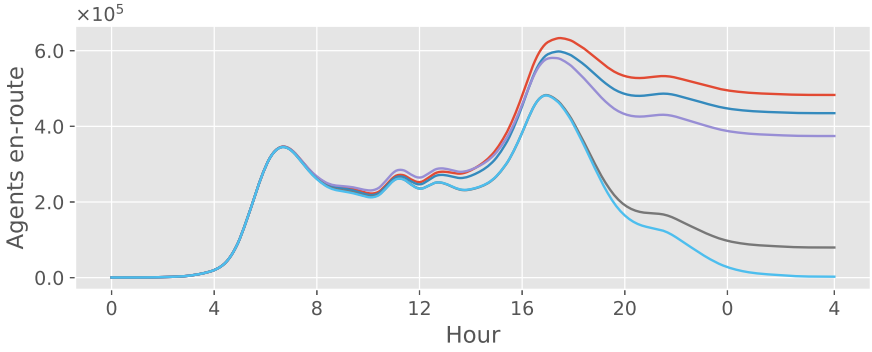
were simulated, public transit was not simulated. For each strategy, 100 iterations of the original scenario were run to converge to equilibrium, where in one iteration a 30-hour long period starting from midnight is simulated with 1-second temporal resolution. A computing node with four Nvidia P100 GPUs (16 GB DRAM on-board) was used to run the simulations.

The *squeeze* strategy requires that the size of the virtual capacity is an input parameter, and Figure 2.23 presents the number of en-route agents during a simulated day for different sizes of virtual capacity. A smaller virtual capacity leads to a larger number of agents becoming stuck during the simulation. After 100 iterations, for small sizes of virtual capacity (4–16) a substantial number of agents are still en-route in the last 6 hours of the simulated 30-hour period; that is, the agents are gridlocked and cannot continue performing their daily plans. On the other hand, for virtual capacities of larger sizes (32 and 64) there are almost identical final results, with no gridlocked agents at the end of the simulated 30-hour period; the relative difference is less than 8%. Therefore, virtual capacities between 32 and 64 completely resolve gridlocks in a large-scale simulation. Thus a virtual capacity of 64 was used below to compare the performance of the three gridlock resolution strategies.

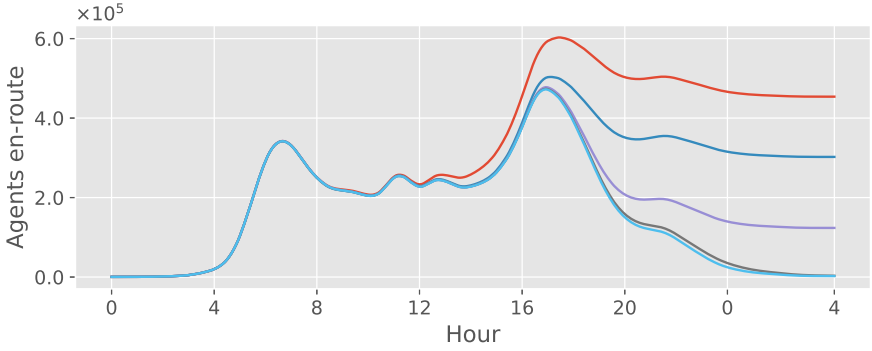
The impact of proposed gridlock resolution strategies on simulation convergence is presented in Figure 2.24. After 25 iterations, all strategies converge to very similar states. It is noteworthy that only the *push-through* strategy shows a faster congestion relaxation after peak hours. This faster convergence is expected as the *push-through* strategy is the least realistic, and can push vehicles over multiple links within a short time. Both the *re-inject* and *squeeze* strategies provide almost identical results. One can also note that the *push-through* strategy provides a good approximation of the final result in just a few iterations. This observation can be used for calibration purposes to save time on a search through a large set of input variables when a large-scale scenario has to be run dozens or hundreds of times.



(a) Iteration 1.

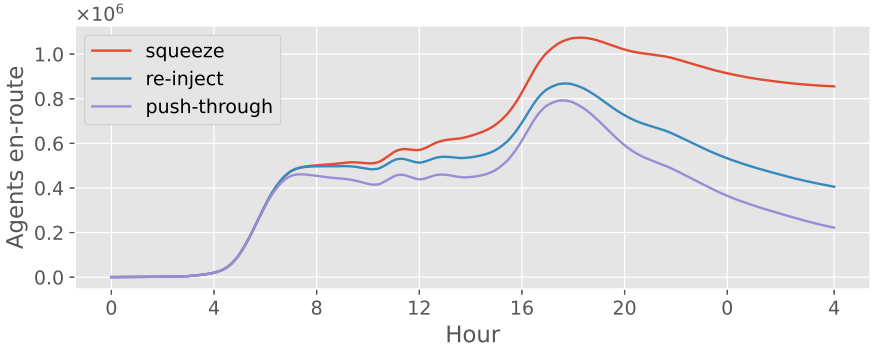


(b) Iteration 25.

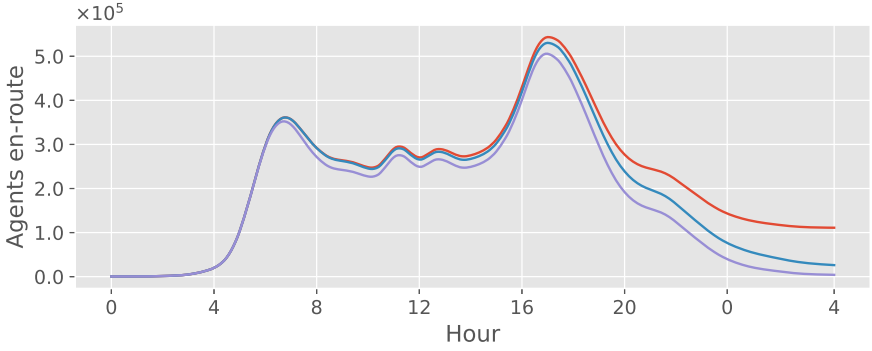


(c) Iteration 100.

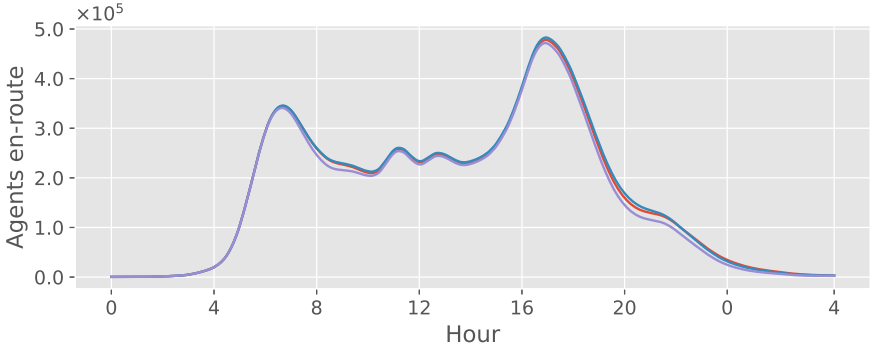
FIGURE 2.23: Agents en-route in Switzerland scenario when using the *squeeze* gridlock resolution strategy with different sizes (4–64) of virtual capacity.



(a) Iteration 1.



(b) Iteration 10.



(c) Iteration 25.

FIGURE 2.24: Agents en-route in Switzerland scenario when using different gridlock resolution strategies.

Table 2.5 gives more insight into the impacts of the different gridlock resolution strategies on the outcomes of the simulation. During simulation, each time an agent gets stuck, the stuck event is recorded. At the first iteration, the *push-through* strategy generates five times more stuck events (44 million) than the *re-inject* strategy (8 million) and four times more than the *squeeze* strategy (10.6 million). This can be explained by the fact that the *push-through* strategy resolves a gridlock much faster, therefore more agents can propagate through the congested network. It is also for this reason that this strategy has a higher vehicle-kilometres travelled (VKT) figure than the other strategies. On the other hand, the *squeeze* strategy has the highest vehicle-hours travelled (VHT) figure, as this strategy is the least efficient (in terms of relaxation speed) for a gridlock resolution, and agents spend more time in congestion. Additionally, the *squeeze* strategy requires about 280–300 MB of GPU DRAM to allocate virtual capacity in the buffers, which is nevertheless acceptable for modern GPUs that already have 80 GB DRAM.

After 100 iterations, the situation changes, and all metrics, except the number of stuck events, are close to each other for all strategies. However, this does not mean that all strategies similarly model the dynamics of individual congested intersections. The *re-inject* strategy has the best runtime (that is, clock time required to simulate one iteration) performance to reach a converged state, because the strategy does not introduce additional atomic operations in GPU kernels. The *squeeze* strategy is about 25% slower due to more scattered memory transactions to larger link buffers with virtual capacity allocated (since all buffers are allocated on a GPU sequentially in a single chunk). Interestingly, for a converged simulation, the *squeeze* and *re-inject* strategies generate twice as many stuck events compared to the *push-through* strategy. The explanation is that a more realistic gridlock resolution relaxes congestion more slowly, thus more vehicles get stuck, whereas a less realistic strategy allows the traffic to flow more smoothly, resulting in fewer stuck events.

Table 2.5. Impact of gridlock resolution strategies on output externalities in the Switzerland scenario. VKT/p and VHT/p are per-person VKT and VHT values, respectively.

Strategy	Stuck events	VKT	VHT	VKT/p	VHT/p	GPU DRAM, GB	Runtime, s
			Iteration 1				
push-through	44 354 624	150 349 601	9 805 151	49.36	3.22	5.09	133
re-inject	7 990 270	141 929 124	7 768 868	46.59	2.55	5.09	218
squeeze	10 598 743	119 122 573	10 275 535	39.22	3.38	5.37	163
			Iteration 100				
push-through	478 856	168 762 000	4 395 740	55.40	1.44	5.10	173
re-inject	971 311	168 601 745	4 423 140	55.35	1.45	5.10	141
squeeze	1 078 083	168 746 345	4 495 997	55.40	1.48	5.39	196

Figure 2.25 shows the location of 10 traffic counting stations in the Zurich area (the largest metropolitan area in Switzerland). Figure 2.26 compares those stations' traffic count data for the year 2017 to the simulation outcomes of the three gridlock resolution strategies. The comparison for a typical working day was made for the morning (07:00–08:00) and evening (17:00–18:00) peak hours. It can be seen that the *push-through* strategy tends to predict slightly higher counts compared to the *re-inject* and *squeeze* strategies. However, on average, the gridlock resolution strategies' predictions capture the trends in the data, and quantitatively the predicted traffic counts have relatively small differences to each other.

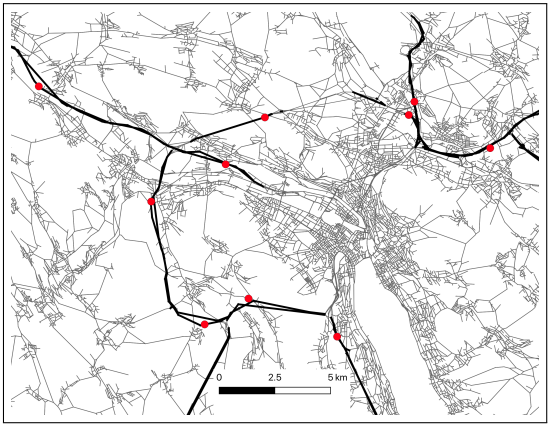
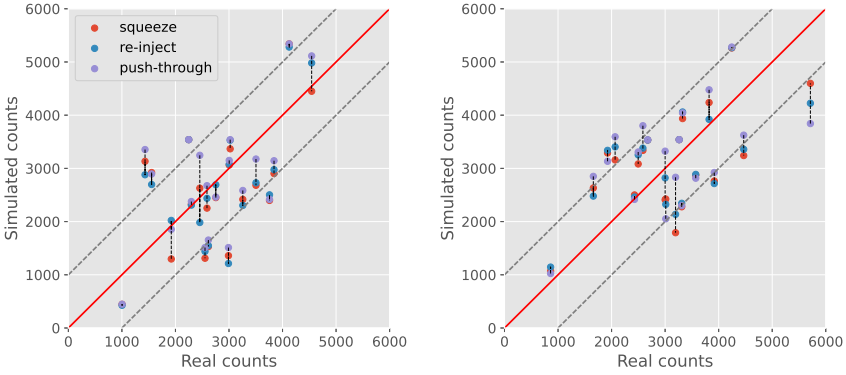


FIGURE 2.25: Locations of the traffic counting stations in the Zurich area used to evaluate the impacts of gridlock resolution strategies.

Overall, the results show that all strategies can efficiently resolve gridlock situations. However, each strategy has advantages and disadvantages, and one has to find a trade-off to apply the strategies in practice: the *push-through* strategy provides faster convergence; the *re-inject* strategy provides a better runtime performance; and the *squeeze* strategy provides a more realistic approach that allows for an easier visualization at the expense of higher memory consumption. A good compromise would be to use a hybrid approach, whereby the *push-through* strategy is applied in the initial iterations, before switching to a more realistic *squeeze* strategy afterwards. This hybrid approach can provide an overall faster convergence of a simulation. The evaluation that is presented here also shows the sensitivity of the performance of a GPU-accelerated traffic simulator when considering practical applications. A gridlock resolution strategy alone reduces the performance of the simulator by up to 25% due to the longer runtime.



(a) Morning peak hour (07:00 – 08:00).

(b) Evening peak hour (17:00 – 18:00).

FIGURE 2.26: Comparison of simulated and real-world traffic counts in the Zurich area using different gridlock resolution strategies.

2.6 VELOCITY MODEL

One of the drawbacks of the original queuing model, as well as the mesoscopic approach in general, is the limited level of detail used for intra-link vehicle dynamics modelling, which defines how the longitudinal speed and the acceleration rate, together with the lateral position of a vehicle, change as it moves along the link. While macroscopic traffic models do not model interactions of individual vehicles, mesoscopic models have only limited interactions like spatial queues. Specifically, in the original queue model, vehicle dynamics are only modelled at intersections, and the movement along a link is supposed to be uniform when a vehicle moves with a certain average speed without acceleration or deceleration.

While intra-link vehicle dynamics can be omitted to model overall traffic flows and congestion patterns in a study area, emerging transport technologies like BEVs require a more accurate estimation of vehicle dynamics in order to model complex physical processes like battery discharge in a more realistic way. For example, an electric battery is sensitive to the acceleration rate applied to the vehicle, and the battery can also recover some energy while the vehicle is driving downhill. Therefore, mesoscopic traffic models shall be adapted for improved estimation of intra-link vehicle dynamics.

2.6.1 *Background*

In microscopic traffic models, a car-following model is used to explain the behaviour of people or AVs driving on the roads while following another car. A vehicle can be classified as following when its speed is constrained by another, preceding, vehicle; otherwise, when not constrained, a vehicle is considered to be at the desired

speed. A following vehicle reacts to the actions of the preceding vehicle in the same lane through the change of its speed. In addition to car-following models, other traffic behavioural sub-models exist, for example, lane-changing models, overtaking models, gap acceptance, ramp merging and so on. As mesoscopic models mostly provide queue-based intra-link interaction between vehicles in single lanes, car-following models can be used as a basis for the extension. According to Olstam and Tapani [186], car-following models can be classified into the following categories depending on the utilized logic:

- **General** models, or the Gazis-Herman-Rothery (GHR) family of models [187], which assume that the acceleration of a following vehicle is proportional to the relative speed and distance to a preceding vehicle.

- **Safety distance**, or collision avoidance models, assume that a vehicle always keeps a safe distance to a preceding vehicle [188, 189]. One of the most widely used models of this category in the existing microscopic traffic simulators is Gipps's model [190], which is based on the desired acceleration and braking rates of the following vehicle. In Gipps's model, a safety distance can be defined as a function of maximum accepted braking rates of both following and preceding vehicles [191]. These models, however, do not consider drivers' perceptions.

- **Psycho-physical**, or action point models, assume that a driver reacts to spacing or relative velocity only when a certain threshold is reached. In contrast, GHR models assume that a driver reacts to any small change in relative velocity, even when the distance to a preceding vehicle is large. The use of thresholds allows the introduction of driving modes, like free driving or emergency braking, when the reaction of a following vehicle is non-linear. Some widely used examples of psycho-physical car-following models include the works of Wiedemann [192, 193] and Fritzsche [194].

There are also some extensions to existing car-following models that make the simulated behaviour more human-driven, for example by using fuzzy logic [195]. As one can see, car-following models can enhance a mobility simulation with realistic human-like behaviour and decision-making on the roads, but at the expense of a more challenging calibration process [196] and increased computational burden. Moreover, it not simply possible to apply the same car-following model to every scenario because of heterogeneity [197] in car-following behaviour (that is, trucks behave differently compared to passenger cars) and transferability problems arising from cultural differences [198]. Therefore, as a mesoscopic model cannot descend to the level of detail of car-following models, another approach is required.

The major problem related to vehicle's speed during its movement in mesoscopic queueing models is the instantaneous change of speed between the links. As no speed change is modelled along a link, a profile looks like a step-wise function where speed limits of the links from a taken route are used. Another problem is

the quality of input data, especially when it comes from crowd-sourced services like OSM where speed limits are not always set, or set improperly. After importing such datasets, and imputing missing speed limits by road type and the number of lanes, there could be situations when speed limits change substantially over a short distance, for example, when exiting from a roundabout or entering a highway from a ramp. Acceleration rate between such network links will be unrealistic, and may lead to biased results when used to model electric battery discharge processes.

2.6.2 *Model implementation*

In order to provide more accurate speed profiles, a mesoscopic velocity tracking model was developed. The model incorporates ideas from car-following models and extends the queuing model to make drivers' decisions more realistic. While in car-following models a vehicle reacts to the behaviour of a preceding vehicle, here, only average time over a network link is calculated to give a better lower-bound estimation of travel time. In case of congestion, the travel time will increase, and a vehicle would not be able to leave the link earlier.

Moreover, in reality, a driver knows more about the environment than just the speed limit of the current link. For example, road signs can indicate changing road conditions, giving a driver time to adapt their speed. A driver may also have previous experience, so that they know the speed limits along the route and can change speed in advance. The developed velocity model uses this fact and gives information to an agent about the next network link, and the agent can decide how to drive along the current link in order to make a comfortable transition to the next link without breaking speed limits or performing emergency braking. In addition to the original model, the speed of each vehicle from the last travelled link is tracked, which requires an extra array of data to be stored in GPU memory.

Like some car-following models, the developed velocity model accounts for multiple driving modes:

- **Normalization mode**, when a driver tries to normalize a vehicle's speed according to the defined speed limit. When a vehicle enters a link with a higher-than-allowed speed, the driver will brake until the limit is reached. Similarly, if the link entry speed is lower, the driver will try to accelerate up to the limit.

- **Cruise mode**, when a vehicle moves at a constant speed and without acceleration. This mode is activated when the entry speed of a vehicle matches the speed limits of the next two links, hence no speed change is required. The cruise mode is also used in the middle of relatively long links, when an agent has time to normalize the speed at the beginning and, if required, before entering the next link.

- **Deceleration mode**, when a driver tries to match the speed limit on the next link by braking.

A detailed algorithm of the velocity model is presented in Figure 2.27. This model is executed in the `ProcessNodes()` kernel each time a vehicle enters a new link, and this also determines the minimum amount of time a vehicle should spend on the link. The model is optional and can be disabled in the configuration file. Here, the following inputs are required to evaluate the speed: v_0 is the velocity of a vehicle when entering a link, v_{lim} is the speed limit of the entered link, v_1 is the speed limit on the downstream link, L is the length of the entered link, and a_{norm} is the acceleration/deceleration rate used by drivers.

In the algorithm, a driver first decides whether the normalization mode shall be activated (lines 2–9) to reduce the entry speed v_0 until the limit v_{lim} or to the closest possible value, and the output is stored in the variable v_{norm}. The variable v_{target} is set to the speed the driver wants to achieve by the end of the link, and this speed is the lowest of the limit v_{lim} of the current link and the limit v_1 of the next link (line 10). After the target speed is set, the driver tries to find an optimal strategy to achieve it.

First, the driver evaluates (line 11) if the length of the link is long enough to enable the cruse mode after the normalization mode is switched off. The cruise mode assumes that a vehicle moves with a constant speed equal to the limit v_{lim}, and, moreover, a driver has enough time to accelerate up to the cruise speed and to brake at the end of the link to meet the target speed v_{target}. This is done by calculating the corresponding distances required for acceleration and braking (lines 12–13), and then a vehicle either gradually changes the speed from entry one to the target (lines 15–16), or goes to the cruise mode (lines 19–20). In the case that the vehicle enters a link too fast or too slow (line 24), the speed changes to the best possible match of the downstream link limits. After the length required for each driving mode is calculated, the minimum travel time can be obtained (lines 26–27). Finally, link exit speed is stored in the v_{exit} variable.

Input: v_0, v_1, v_{lim}, L, a_{norm}
begin
  $[v_{norm}, L_{norm}] \longleftarrow [v_0, 0]$;
  if $v_0 > v_{lim}$ **then**
    $L_{norm} \longleftarrow (v_{lim}^2 - v_0^2)/(-2 \cdot a_{norm})$;
    if $L_{norm} > L$ **then**
      $v_{norm} \longleftarrow \sqrt{2 \cdot L \cdot (-a_{norm}) + v_0^2}$;
      $L_{norm} \longleftarrow L$;
    end
  end
  $[v_{target}, L_{dec}] \longleftarrow [min(v_{lim}, v_1), 0]$;
  if $L_{norm} < L$ **then**
    $L_{acc} \longleftarrow (v_{lim}^2 - v_{norm}^2)/(2 \cdot a_{norm})$;
    $L_{dec} \longleftarrow \left|(v_{target}^2 - v_{lim}^2)/(2 \cdot a_{norm})\right|$;
    if $L_{acc} + L_{dec} > L - L_{norm}$ **then**
      $v_{exit} \longleftarrow min(\sqrt{2 \cdot a_{norm} \cdot (L - L_{norm}) + v_{norm}^2}, v_{target})$;
      $L_{dec} \longleftarrow \left|(v_{exit}^2 - v_{norm}^2)/(2 \cdot a_{norm})\right|$;
    end
    else
      $[v_{exit}, v_{norm}] \longleftarrow [v_{target}, v_{lim}]$;
      $L_{norm} \longleftarrow max(L_{norm}, L_{acc})$;
    end
  end
  else
    $v_{exit} \longleftarrow v_{norm}$;
  end
  $L_{cruise} \longleftarrow L - L_{norm} - L_{dec}$;
  return $|v_{norm} - v_0| / a_{norm} + L_{cruise}/v_{lim} + |v_{exit} - v_{norm}| / a_{norm}$;
end

FIGURE 2.27: Algorithm of the velocity tracking model on the GPU used to improve the modelling of intra-link vehicle dynamics.

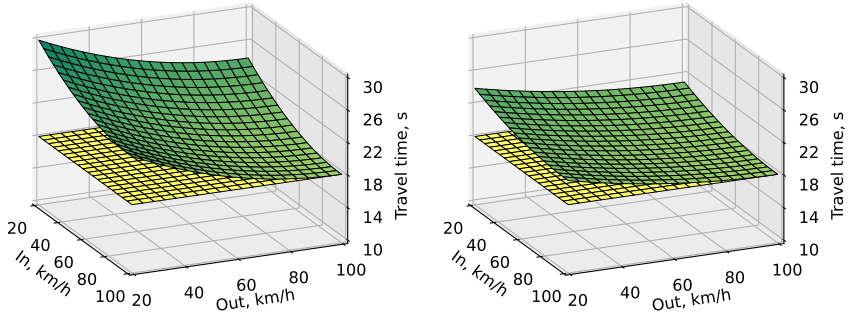
The value for a_{norm} defines comfortable acceleration and braking rates, and can be taken from the literature or by using other, personal assumptions. For example, Proctor et al. [199] used a 1987 Lincoln Mark VII LSC car to test the acceleration rates of passenger cars, and the results showed the acceleration varies from about 1.5 m/s^2 to about 4.12 m/s^2. Hoberock [200] did a survey of 11 studies on passenger comfort

due to longitudinal motion, and the results showed that in public mass transport non-emergency accelerations from 1.08 m/s^2 to 1.47 m/s^2 are in the acceptable range for most studies, although larger values could also be accepted. Values larger than 2.94 m/s^2 are unlikely to be accepted for most public transport. Panwai and Dia [201] collected radar data in Stuttgart (Germany), during an afternoon peak hour from a single lane under stop-and-go conditions. They found that vehicles tend to have acceleration rates in the range from -4 m/s^2 to around 2 m/s^2. Wee et al. [202] used GPS data from Singapore to calibrate simulated vehicle speeds along signalized arterial roads using the microscopic model of PARAMICS. The authors found that an average acceleration rate of 2.05 m/s^2 allowed it to fit field datasets with high accuracy.

Based on the above-mentioned literature, values for a_{norm} can be set in the range of $1 - 2$ m/s^2. Moreover, as many studies show that there is no exact value for comfortable acceleration as people perceive longitudinal forces differently, a per-agent value for a_{norm} can be set based on personal attributes. This will require an additional array to be allocated in GPU memory.

The value of a_{norm} defines the response rate at which a driver wants to recover the difference between entry speed and the speed limit of a link, as well as the difference between the limit of the current link and the limit of the downstream link. In other words, it can be treated as "driving aggressiveness". While this parameter defines which driving modes will be enabled on a link and for how long, it also impacts the total link travel time in free-flow conditions. Figure 2.28 shows the distribution of travel times along a 500 m link with a speed limit of 100 km/h, for different combinations of the entry speed (in speed) and the speed limit set for the downstream link (out speed). Two types of behaviour were modelled: relaxed behaviour when $a_{norm} = 1.5$ m/s^2, and more aggressive behaviour when $a_{norm} = 3.0$ m/s^s.

The horizontal plane around the z-value of 18 is the output of the original model, and the travel time depends neither on entry speed nor the speed limit of a downstream link. The more aggressive a driver, the closer the surface generated by the velocity model to the original plane. This can be simply explained by the fact that a driver can accelerate and brake faster, thus approaching the instantaneous change in speed from the original model. The difference in travel time from the original model, in the case of aggressive driving, is relatively minor and results in about four additional seconds (about 22% increase); for the case of lower a_{norm}, the difference can reach almost 12 seconds, or about a 67% increase. However, such extreme cases, when high-speed and slow links are on both endpoints, should rarely occur in reality or simulations, and in most situations only minor differences in the range of one to two seconds shall be observed.



(a) Relaxed driving style with acceleration rate $a_{norm} = 1.5\ m/s^2$.

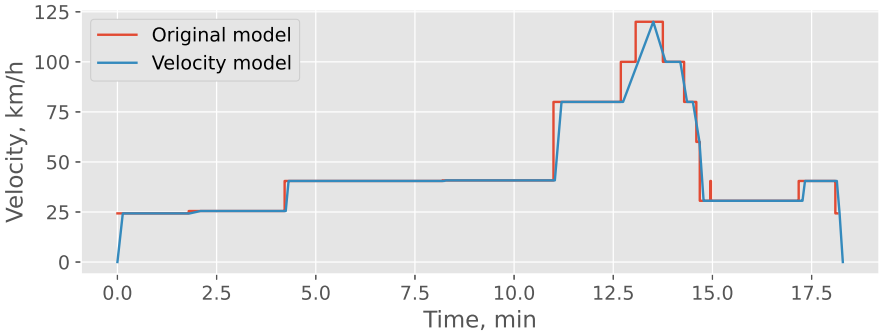
(b) Aggressive driving behaviour with acceleration rate $a_{norm} = 3.0\ m/s^2$.

FIGURE 2.28: Distribution of simulated travel times along a link depending on entry speed and the speed limit of a downstream link.

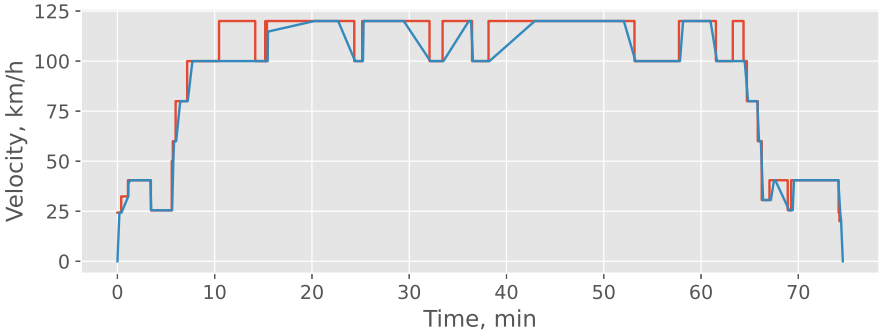
While the travel time does not change significantly with the introduction of the velocity model, the main impact can be observed in tracking the velocity of individual vehicles. Figure 2.29a shows the recorded speed profile of a simulated agent when driving in the city of Zurich, Switzerland, from the Wiedikon area to Dübendorf, which is located on the other side of the city border. Here, free-flow conditions were modelled without any congestion. Most of the time the agent drives in the city area with speed limits of no more than 40 km/h, with only a brief period on the highway, before reaching the destination area with lower speed limits again. Figure 2.29b shows the same output, but for another agent travelling from Bern to Zurich, hence making a long-distance inter-regional trip for which most of the time was spent on a highway.

First, one can see that the original model produces a step-wise speed profile, with instantaneous changes of speed. In contrast, the developed velocity model produces a more realistic profile, with gradual changes of speed. This is especially visible in the beginning and at the end of the trips, when in realistic conditions a vehicle should have zero speed. Second, some sharp peaks from the original model are smoothed out, for example around minute 15 in Figure 2.29a and around minute 63 in Figure 2.29b. These peaks were produced because routes have some short parts with higher speed limits on the links, and the original model simply changes up and down. The velocity model, instead, can gradually change, when possible, speed and switch a driver to the cruise mode followed by gradual decrease of the speed back to the lower limits. Moreover, on short links drivers tend not to vary significantly.

This velocity smoothing behaviour results in one drawback to the velocity model, in that it can lead to lower average speeds on some links. For example, around minute 10 in the inter-regional trip, there is a 7 km highway link with a speed limit



(a) City trip from Wiedikon to Dübendorf in the canton of Zurich, Switzerland.



(b) Intercity trip from Bern to Zurich, Switzerland.

FIGURE 2.29: Simulated speed profiles of a vehicle when driving in different traffic conditions.

of 120 km/h, however the model produces the average speed of 100 km/h only. The reason for such behaviour is the following: as the high-speed link has lower-speed entry and exit limits, and a driver has enough time to optimize his/her behaviour, the vehicle spends most of its time in cruise mode, before braking to the same speed as at the entry. As intra-link dynamics are not captured and not recorded, this part of the behaviour is lost, making no difference between entry and exit speeds. This will mean a lower power consumption if a BEV has been driving this route, even though the vehicle can regenerate some energy while braking. The solution for this problem is to slightly modify the input network to split long links into parts, so that the first and last parts will provide acceleration and braking behaviour, respectively, while other parts will capture the real speed limit of the original link.

2.7 VEHICLES

Many initial studies on agent-based traffic simulations did not distinguish between people and vehicles and used person-vehicle equivalents, when vehicle attributes are attached directly to a person. This approach simplifies the representation of the data used in a simulation, as well as reducing the complexity of the models by effectively reducing the number of modelled agents. On the other hand, the use of person-vehicle equivalents creates some difficulties in modelling complex and multi-modal scenarios, for example simulating non-car modes of transport like public transit or taxis where passengers are be accounted separately, or having agents with multiple cars in their daily-activity plans with detours. Moreover, with the development of emerging technologies like BEVs, it becomes of increasing importance to model the state of vehicles individually as the state impacts the behaviour of the agents.

In GEMSim, vehicles are modelled individually when needed through the vehicle index, specified in the base travel leg data structure in Listing 2.6. When an agent does not have an associated vehicle for a travel leg, the state of the vehicle is not modelled and the parameters of a generic vehicle are used in simulation. This dynamic approach allows a reduced computational burden when it is not needed. Vehicles are always grouped in fleets, and separate fleets can be specified in multiple parts of input data, for example each taxi operator or public transit agency has its own fleet.

The description of a fleet is twofold: (i) models of vehicles with their corresponding attributes, and (ii) vehicles of certain models used by the agents. This description allows the separation of the properties of the vehicles from their states, where a state defines the values of the properties. Second, it allows the optimization of the amount of input data, as many agents use the same models of vehicles but each model only has to be described once. Third, this approach also simplifies the random assignment of vehicles to agents based on the available statistical data of car markets. Each vehicle type, used to describe a specific vehicle model, has a set of attributes: fuel type (energy source), physical dimensions and mass, frontal area and drag coefficient, available space for passengers, etc. In this set, some attributes are mandatory and some not, depending on the fuel technology of a vehicle. For example, the drag coefficient and mass are required for BEVs, while for ICEVs these attributes are optional and do not affect the modelling process.

One of the main purposes for the modelling of individual vehicles is fuel consumption tracking. For fossil fuel vehicles this allows the estimation of externalities like air pollution, while for electric vehicles it allows the assessment of impacts on charging infrastructure and distribution grids. Such a fuel consumption tracking subsystem was implemented in GEMSim. It comprises three main components:

- tracking of fuel consumption of individual vehicles;
- refuelling behaviour;

- refuelling infrastructure.

Depending on the fuel types, GEMSim supports the following classes of vehicles:

- **ICEV**. Vehicles with internal combustion engines, using liquid fuels like petrol or diesel.
- **BEV**. Vehicles with electric batteries, using electricity from batteries as a source of energy.
- **Wired**. Vehicles which are attached to wires or other types of electrical links, use electricity coming from these links as a source of energy.

Each of the classes defines a set of additional attributes specific to a given vehicle type, and the state of each individual vehicle is based on the type of vehicle used.

In general, vehicle types can be split in two categories by the level of detail used to model them in GEMSim: non-BEVs and BEVs. Non-BEVs are modelled in a simplified way as these vehicles are already well studied, and they are more predictable in terms of fuel consumption and the impacts on infrastructure and society through negative externalities. In contrast, BEVs have only started penetrating markets, hence, this class of vehicles gains more attention from researchers, society and policy-makers, and there are multiple reasons why BEVs require more accurate modelling.

First, the state of a BEV is more sensitive to the environment. For example, driving cycle, weather conditions and elevation profile have a strong impact on the battery's SoC. At the same time, the typical driving range of a BEV is shorter compared to modern ICEVs, while the recharging takes longer (up to several hours, depending on the type of the charger used). These factors affect the behaviour of agents with BEVs for whom the refuelling process is more critical in daily-activity planning, especially considering the scarcity of charging infrastructure. Second, the demand for charging infrastructure, and the consequent impact on local electric distribution grids, is more complex to predict. For example, one can refuel an ICEV in a few minutes, while recharging of a BEV can vary from minutes to hours, depending on SoC, charging point and the BEV's capabilities. Moreover, as many countries set goals to reduce the number of private non-BEVs in order to make positive contributions into the reduction of global CO_2 emissions, there is a high probability that the transition to BEVs will put more BEVs on the roads in upcoming years. Therefore, more accurate modelling of BEVs and related infrastructure is required for better understanding of the mobility transition process. As one of the goals of this thesis was to develop a large-scale mobility model with the focus on emerging technologies, BEVs are modelled with a high level of detail, accounting for their specific physical properties and surrounding environmental conditions, while for other types of vehicles simplified models based on per-distance average fuel consumption are used. This, however, does not prevent possible future implementations of fuel consumption models for non-BEVs, as the framework has a flexible and modular architecture.

Refuelling behaviour is modelled at two points of the simulation loop. The first point is during the simulation, when a vehicle runs out of fuel and the driving agent can make a decision where and how to refuel. This behaviour is only supported for BEVs, while for other vehicle classes it is assumed that refuelling activities are performed on-the-go and do not affect the daily-activity planning of the agents. The second refuelling point happens between simulated iterations, and it defines the initial simulation conditions in the scenario. For example, fuel levels can be either distributed according to a certain law, or each vehicle can be fully fuelled. This decision is performed through the special fleet refuelling handlers which are registered in the main simulation loop, and automatically executed upon the end of an iteration. Some default implementations are available: for example, refuel to maximum level, or use the same level as at the beginning of the previous iteration. More complex decision-making models can easily be integrated into the simulator. Refuelling infrastructure, for the same reasons as refuelling behaviour, is only modelled for BEVs, as described in Subsection 2.7.2.

The main structure of a vehicle fleet on a GPU is presented in Listing 2.7. The structure contains the static attributes of vehicle models in the array `vehicles` of base structures, while per-vehicle dynamic states, namely fuel level and occupancy, are stored separately in the arrays `fuel_level` and `space_occupied`, respectively. The number of vehicles `vehicle_count` is used to iterate through the fleet.

```
struct GpuFleet {
        // Per-vehicle data
        GpuBaseVehicle * vehicles       []; // Vehicles

        // Per-vehicle state
        float            fuel_level     []; // Fuel level
        int32_t          space_occupied []; // Occupied space, pax

        int32_t          vehicle_count;     // Vehicle count
};
```

LISTING 2.7: Structure of a vehicle fleet on a GPU storing vehicle models and state of each vehicle.

Here, an approach similar to storing travel legs in GPU memory was used, when the base vehicle structure `GpuBaseVehicle` contains common attributes of a vehicle, and this structure is nested into a more specialized structure that represents a specific class of vehicles. In contrast to the AoS approach used for vehicle attributes, SoA is used to store vehicle states. The reason is that when a GPU thread needs to update the state of a vehicle (typically, when a vehicle changes a network link), many attributes are required as inputs to algorithms, and at the same time, this update event is scattered across the vehicles, making it difficult to perform coalesced memory access. Therefore, storing per-vehicle attributes as AoS allows for a GPU to cache per-thread data efficiently. Vehicle states, however, use SoA because this data

can be accessed from the CPU side, for example to model refuelling behaviour, and it is also more convenient to export data for further post-processing and analysis as part of simulation outputs. GPU memory for attributes is allocated in a single chunk and then partitioned, similar to travel legs.

The base vehicle structure in GPU memory is presented in Listing 2.8. It contains the common attributes of vehicle models, and these attributes are used as inputs to update the states of vehicles during the simulation. The `veh_type` field indicates the class of a vehicle and is used to promote the base structure to a specialized one.

```
struct GpuBaseVehicle {
        int32_t vehicle_id;       // Vehicle ID
        float   mass;             // Mass, kg
        float   drag_coef;        // Drag coefficient
        float   frontal_area;     // Frontal area, m^2
        int16_t space_available; // Available space, pax
        int16_t veh_type;         // Vehicle type
};
```

Listing 2.8: Base structure of a vehicle on a GPU with common attributes used to update the state.

2.7.1 *Non-BEVs*

ICEVs have tank volume and average fuel consumption per driven kilometre specified as class-specific attributes. ICEVs are modelled in a simple way, when fuel consumption F_{ICEV} is directly proportional to the driven distance D_{km}:

$$F_{ICEV} = D_{km} \cdot f_{ICEV} \tag{2.24}$$

where f_{icev} is the average fuel consumption per kilometre for this vehicle type. The structure of an ICEV in GPU memory, in addition to the base vehicle structure, contains tank capacity and fuel consumption f_{ICE} of the vehicle.

Vehicles that use wires or special links to get electricity as a source of energy, and are modelled in a way similar to ICEVs, when the energy consumption E_{wired} is directly proportional to the driven time T_s:

$$E_{wired} = T_s \cdot p_{wired} \tag{2.25}$$

where p_{wired} is the average power consumption for this vehicle type, and it is the only class-specific attributed required as an input. While p_{wired} is specified through class-specific attributes (as for ICEVs), it does not require any limits for fuel storage, so the state of such a vehicle type contains total consumed energy stored instead of the fuel level. Wired vehicles are mostly intended for public transit simulations to estimate rough energy consumption figures for trams and trolleybuses. The structure of a wired vehicle in GPU memory, in addition to the base vehicle structure, contains average power consumption p_{wired} of the vehicle.

2.7.2 *BEVs*

BEVs have battery capacity, supported plugs and charging power specified as mandatory class-specific attributes, while the availability of a heat pump is optional. The state of this vehicle class keeps track of how much energy left in the battery. The energy consumption model is based on the work of Fiori et al. [203]. In the first step, the power at wheels is calculated at the moment of simulation time t:

$$P_{wheels}(t) = [m \cdot a(t) + m \cdot g \cdot \cos(\theta) \cdot C_r \cdot (c_1 \cdot v(t) + c_2) + \frac{1}{2} \cdot \rho_{air} \cdot A_f \cdot C_D \cdot v^2(t) + m \cdot g \cdot \sin(\theta)] \cdot v(t) \quad (2.26)$$

where m is the gross mass of the vehicle in kg; $a(t)$ is the acceleration of the vehicle in m/s^2, $g = 9.8066\ m/s^2$ is the gravitational acceleration constant, θ is the road slope; $C_r = 0.00175$, $c_1 = 0.0328$ and $c_2 = 4.575$ are the rolling resistance parameters coming from the road surface type, road condition, and vehicle tire type; $\rho_{air} = 1.2256\ kg/m^3$ is the air mass density at sea level at $15°C$; A_f is the frontal area of the vehicle in m^2; C_D is the aerodynamic drag coefficient of the vehicle; and $v(t)$ is the speed of the vehicle in m/s.

The value of m comprises vehicle empty mass from the input data plus the total mass of passengers in the vehicle using the average human body mass of 70 kg. This allows to quantify the impacts of occupancy on public transit power consumption. The values of $\cos(\theta)$ and $\sin(\theta)$ are calculated for each network link in advance using the digital elevation model of the area, but this is only possible when the z coordinate for each of the network nodes is provided in the input; these values are stored in GPU memory separately in per-link arrays. The speed value $v(t)$ is calculated as link distance divided by the link travel time of the vehicle, while the acceleration $a(t)$ is calculated based on the entry and leave speed values for the link using the velocity model from Section 2.6. The values of A_f and C_D are taken from the input description of the vehicle.

In the second step, the power at the electric motor is calculated by accounting for the losses in the powertrain:

$$P_{emotor}(t) = P_{wheels}(t) \cdot \eta_{driveline} \cdot \eta_{emotor} \quad (2.27)$$

where $\eta_{driveline} = 0.92$ is the driveline efficiency and $\eta_{emotor} = 0.91$ is the efficiency of the electric motor.

Next, the net electric power is calculated depending on whether the power at the electric motor is negative or positive. In the case of regenerative braking, when $P_{emotor} < 0$, the net electric power is defined as follows:

$$P_{emotor,net} = P_{emotor} \cdot \eta_{battery} \cdot \eta_{rb} \quad (2.28)$$

where $\eta_{battery} = 0.90$ is the efficiency of the battery, and the efficiency of regenerative braking η_{rb} is defined as follows:

$$\eta_{rb} = \begin{cases} \left(\exp^{\frac{0.0411}{|a(t)|}} \right)^{-1} & \forall a(t) < 0 \\ 1 & \forall a(t) \geq 0 \end{cases} \tag{2.29}$$

where negative acceleration means that a vehicle regenerates energy.

In addition to the powertrain, the auxiliary power P_{aux} consumption is modelled depending on the availability of a heat pump and the ambient temperature. Figure 2.30 shows the power consumption curves used to model auxiliary systems [34]. The ambient temperature comes from a dedicated GEMSim service that can define the variation in temperature on a per-link basis during the day, as the scale of a simulation can be country-wide, covering different climate zones. Otherwise, one can specify a single global temperature value for the scenario.

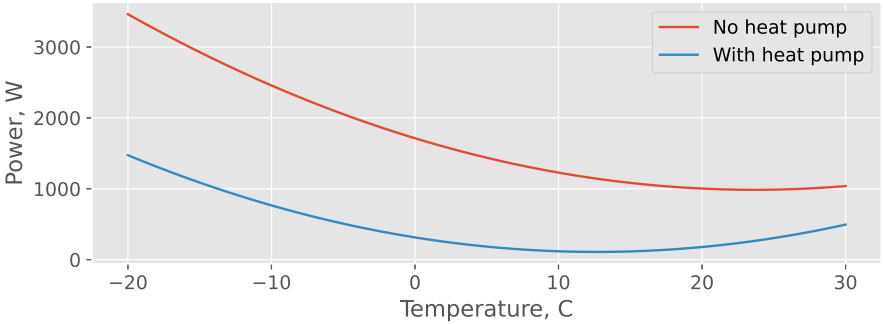


FIGURE 2.30: Auxiliary power consumption of a BEV depending on heat pump availability and ambient temperature.

Finally, the total power consumption is quantified, and the battery capacity is updated based on the travel time of the agent along the network link:

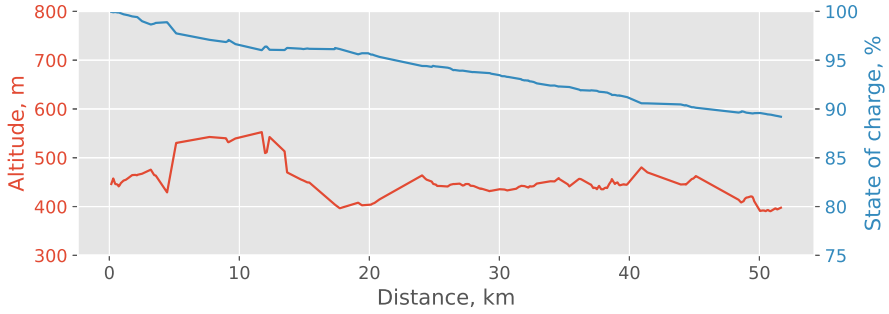
$$P_{total}(t) = P_{emotor,net} + P_{aux} \tag{2.30}$$

Charging infrastructure is critical for the realistic simulation of BEVs as it impacts the behaviour of people in terms of which locations for activities they select. It can also generate additional travel demand for charging activities. To support the simulation of scenarios involving the charging behaviour of the agents, a specification of BEV charging infrastructure as part of input supply for GEMSim was implemented. The decision-making process for charging of BEVs is performed at the host side as it includes spatial lookup and other complex algorithms not particularly suitable for GPUs. The infrastructure comprises a set of geo-referenced charging stations, where each of these stations has one or more charging slots

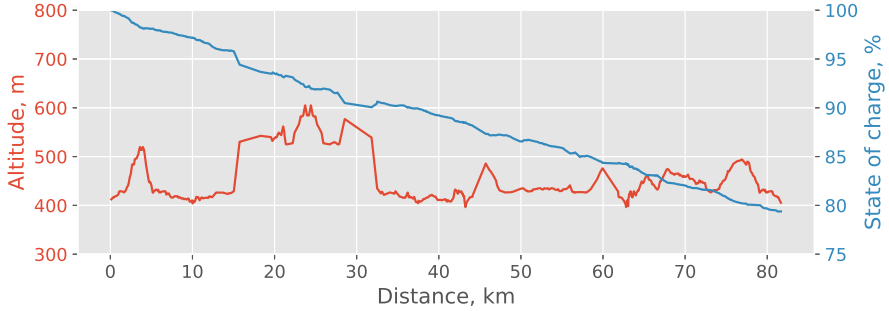
with defined charging type technology (AC or DC), compatible plug types and the maximum output charging power. This input charging infrastructure is used by simulation algorithms to find a compatible charger whenever agents decide to charge their BEVs. One can also implement custom decision-making algorithms on how to agents look for chargers except using distance and plug compatibility, for example, based on dynamic pricing options or parking.

Figures 2.31–2.33 show the change of SoC against altitude, speed and acceleration of three simulated trips of a Tesla Model S vehicle with a battery capacity of 100 kWh in the Zurich area. First of all, steeper altitude profiles have a significant impact on the battery discharge, while downward slopes help to regenerate energy. Higher velocity, as expected, leads to increased power consumption, and spikes in acceleration cause more intensive power demand. As the acceleration of the vehicle has to be calculated for the estimation of power consumption, the velocity model is always enabled whenever the fuel tracking model is used in a scenario. The fuel tracking model is also optional and can be disabled in the scenario configuration file.

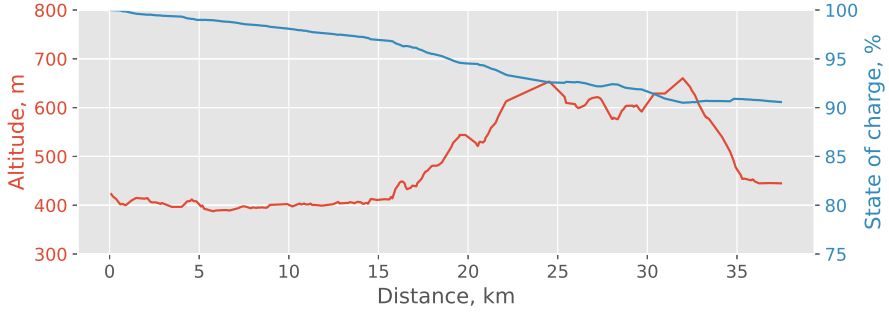
The structure of a BEV in GPU memory, in addition to the vehicle base structure, contains battery capacity and a flag that indicates the availability of a heat pump.



(a) Trip 1.



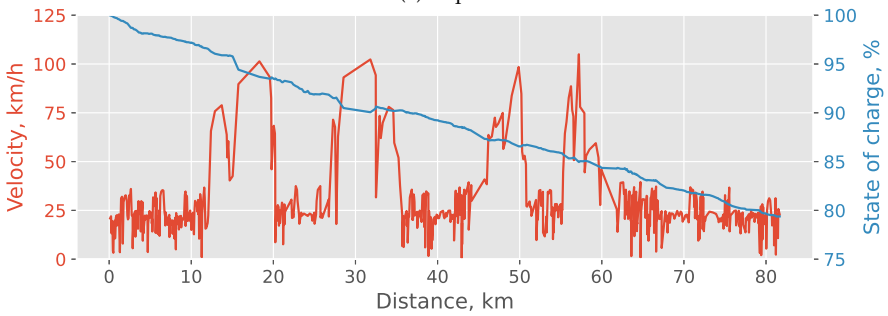
(b) Trip 2.



(c) Trip 3.

FIGURE 2.31: Variation in SoC of a simulated BEV (Tesla Model S) against altitude profile, for multiple trips in the Zurich area.

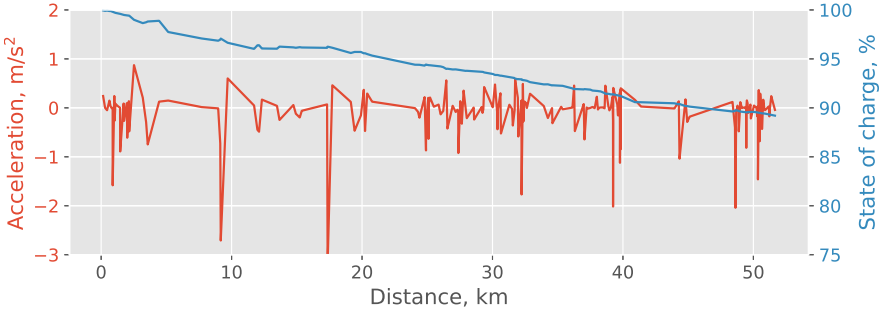
(a) Trip 1.



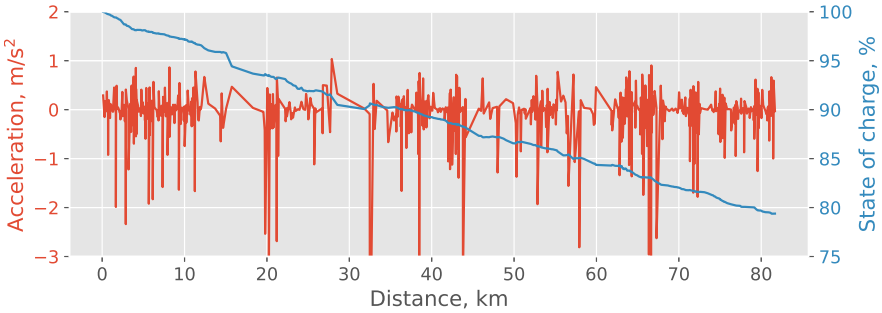
(b) Trip 2.

(c) Trip 3.

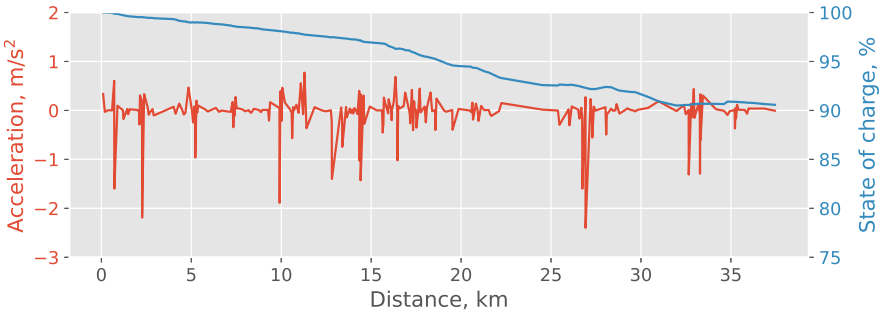
FIGURE 2.32: Variation in SoC of a simulated BEV (Tesla Model S) against speed profile, for multiple trips in the Zurich area.



(a) Trip 1.



(b) Trip 2.



(c) Trip 3.

FIGURE 2.33: Variation in SoC of a simulated BEV (Tesla Model S) against acceleration profile, for multiple trips in the Zurich area.

2.8 HETEROGENEOUS HARDWARE

One of the ideas behind this thesis was not only to develop the methodology for a GPU-accelerated mobility simulator, but to do so in a generalized way so that the technology could be transferred to other parallel hardware, such as modern many-core CPUs. This required two steps to be implemented: (i) memory and data management for specific hardware, and (ii) a hardware abstraction layer for the simulator such that it does not deal with heterogeneous hardware directly. The first step was needed because of separate memory space used at host and on a device like a GPU. While there are developments in a coherent memory space provided for both CPUs and other attached accelerators like GPUs and FPGAs, a more general approach will make the model compatible with a broader range of devices. The second step allowed the development of mobility simulation models without dealing with hardware specifics, thus making models hardware-agnostic. Separation of mobility models from hardware also reduced the development effort as algorithms only need to be implemented once, after which they can be scaled onto available hardware automatically.

This section describes how the data is managed on the host and the device, and also introduces a hardware abstraction layer that permits running the same mobility model on GPUs and many-core CPUs.

2.8.1 *Introduction*

A typical approach to supporting the use of heterogeneous hardware in mobility simulations is to apply specially designed frameworks such as OpenCL [204], which allow one to run the same model on CPUs, GPUs, FPGAs or other accelerators. OpenCL provides a way of writing functions (kernels), which are then executed in parallel on the supported hardware. Furthermore, OpenCL automatically maps kernels on available computing cores (elements). However, while OpenCL is portable across different hardware, there are several limitations. First, hardware vendors must provide the required drivers for OpenCL [205]. Second, the performance of a model can vary a lot across the hardware of different vendors [98], and some vendors may only provide a partial implementation of the OpenCL standard. Third, 1.x, the most widely used version of OpenCL, does not support raw device pointers but rather memory buffers [206]. Thus, it is not a trivial task to use complex and linked data structures, such as trees, on an OpenCL device. Last but not least, in many cases, OpenCL requires additional tuning to achieve optimal performance on specific hardware [207–209], although for some applications auto-tuning methods have been developed [210, 211].

A different approach to facilitating the use of heterogeneous hardware in mobility simulations, applied in this thesis, is to combine parallel computing with hardware accelerators in such a way that a model can be run on either CPU or GPU with minimum effort in development and without using OpenCL.

It is evident that the already discussed data structures are well suited for CPUs in terms of runtime performance. First, the data is compact and organized sequentially in memory (without cross-pointers between nodes and links). A CPU can exploit cache locality when multiple links or nodes are processed one-by-one. Second, multiple CPU cores can process links and nodes in parallel, as with GPU threads, but in order to avoid false sharing (that is, when multiple cores write into the same cache line), each core must process a chunk of items instead of one item as done by a GPU thread. To run the same traffic model around the same data structures on both CPU and GPU, the following abstractions were made:

- **Backend.** The aim of a backend is to abstract hardware management and parallel code execution from the simulator. A backend is used to initialize the hardware; to execute a simulation step in parallel; and to provide hardware-optimized functions such as atomic operations.

- **Data binder.** The aim of a data binder is to abstract data layer management from specific hardware. On the host side, the data layer was implemented using classes, including complex data structures such as collections and trees. However, the host structures do not provide optimal performance because the traffic model has to iterate over a large number of links, nodes and agents, and each object in a collection can be allocated in a random memory location. A data binder maps host objects onto cache-friendly structures using memory functions from the backend.

The main reason why GPU-optimized data structures are not widely used in developed CPU-based models is the complexity of the code development and maintenance, especially for large projects. To simplify the development process, typically object-oriented paradigms are used, complemented with object-oriented programming languages. Therefore, while there is no surprise about runtime performance drop when using rapid development methods, this is often accepted as a cost of having a simpler and faster approach to delivering software. On the other hand, already having a GPU-accelerated model implemented means that one can potentially benefit from the possibility of running the same model on many-core CPUs more efficiently. The proposed approach for running a mobility model on both CPUs and GPUs aims at minimizing the cost of development of such heterogeneous software.

2.8.2 *Background*

Xiao et al. [96] studied the performance improvement when parts of a CPU-based microscopic simulator are offloaded to a GPU using the OpenCL framework. Singapore's road network was simulated, and the generation rate of agents varied from 8 to 400 per simulated second. A fully offloaded GPU-based version had a speed-up factor of 28.7 compared to a sequential CPU-based version in conditions of low-intensity traffic, and up to a 14.8 speed-up factor in conditions of high-intensity traffic. On a multi-core CPU (four physical cores) a speed-up factor of

6.7 was achieved. The authors noted that a fully OpenCL-based implementation was challenging to develop, had limited maintainability and extensibility of the model, and that many commonly used data structures had to be re-implemented for hardware accelerators. In later work, Xiao et al. [97] developed an OpenCL-based microscopic agent-based simulation model that was applied to small-scale (16 384 agents on a single four-lane road) traffic simulations executed on CPUs, GPUs and FPGAs. It was demonstrated that an FPGA ran 24.35 times faster than a CPU core, and 8.9 times faster than a GPU. Rajf and Potuzak [98] compared the runtimes of two microscopic traffic models on CPUs and GPUs that used the same algorithms. Using artificial grid networks and up to 250 000 initial vehicles, a speed-up of up to a factor of 12.4 was shown for the GPU-based model compared to the CPU-based multi-threaded (six physical cores) model. While it was intended to keep both models as similar as possible, the CPU-based and GPU-based versions were respectively implemented with the .NET and the OpenCL frameworks. Interestingly, the authors showed that the GPU model ran significantly slower on AMD than Nvidia hardware; this difference may have been because of the less efficient implementation of the OpenCL driver. While other studies on hardware-accelerated traffic models [75, 79, 88, 99] have been presented, these are only single-threaded implementations for CPU, and often have limited functionality.

Nevertheless, the above-mentioned literature shows that runtime is improved when hardware accelerators are used. However: (i) these previous works rely on the OpenCL framework and its attendant limitations; (ii) none of these previous works demonstrate scalability and improved performance of CPU-based versions on many-core CPUs (of up to six physical CPU cores only); and (iii) none of these previous works demonstrate that multi-modal traffic can be run. Therefore, the following research question is still open: to what extent can modern many-core CPUs compete with GPUs in agent-based mobility simulations when the code is optimized for execution on heterogeneous hardware?

2.8.3 *Data binder*

Data binding is one of the important contributions of this work to the more widely known structure of the simulation loop. This step is either missing or not considered in previous literature, which is surprising when one considers that input data is somehow given and can be put on a GPU easily. For small-scale scenarios, indeed, the transfer of input and output data between the host and a device does not have a significant impact on the overall simulation time and can be omitted in considerations of the performance, but for large-scale scenarios with the feedback loop, data transfers to or from a GPU can become a bottleneck. The bottleneck happens due to the need to use data structures that provide less scattered or even a coalesced access to memory for GPU threads. Therefore, as data structures on the host are different from those used on a GPU, a conversion procedure is required to exchange the data between the host and the GPU.

The data binder is a software unit (class) that incorporates logic on how a certain part of the host data must be placed on a GPU in order to maximize the overall performance. A binder (i) manages GPU-optimized structures that are suited for faster code execution, (ii) allocates memory on a GPU and partitions memory according to a certain layout, (iii) transforms host data into GPU structures, (iv) copies data to partitioned GPU memory, and (v) finally releases the allocated resources. Another important role of a binder is to provide bidirectional mapping of the data from host objects to corresponding GPU objects and vice versa. Nearly all objects (that is, network links and nodes, persons and vehicles) of the input data have unique IDs that are used as short, quick references. On the host side, data structures like key-value associative arrays can be used to quickly access a certain object using its ID; this is not the case for a GPU, due to its limitations in dynamic memory allocation and data structures with random memory access. Instead, mapping is used so that the GPU code operates only with predefined array indices to improve the utilization of memory bandwidth.

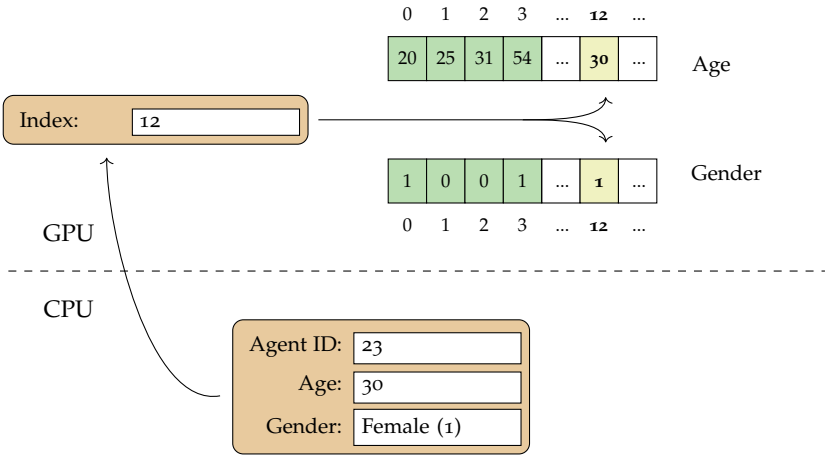


FIGURE 2.34: Data mapping between CPU and GPU address spaces in GEMSim.

Figure 2.34 shows an example of data mapping between the host and the GPU. Here, at the host side, a program has an object (class instance) in memory with the attributes of ID, age and gender, which are located successively in memory. On the other hand, the GPU stores age and gender attributes in separate arrays, and uses an index instead of ID to access the attributes in constant time. Constant time memory access is achieved by avoiding a non-linear procedure that looks up an object by its ID. Moreover, for optimization purposes, the order of the objects in GPU memory may be completely different from the order on the host. This mapping is more complex for composite objects, where each item may have local indexing within a group and global indexing within a composite object. For example, agents

are grouped in sets called populations, and each population object maintains the agents independently from other sets. One set may represent input synthetic agents, and another set may represent artificially created drivers of public transit vehicles. However, as was shown in Subsection 2.3.5, the attributes of all simulated agents are stored together in separate arrays (one array per attribute). To deal with such a composite structure, a binder stores information for hierarchical indexing. The global index $I_{global,i}$ of the i-th object from a set of items on a GPU is defined as follows:

$$I_{global,i} = I_{group,i} + I_{local,i} \tag{2.31}$$

where $I_{group,i}$ is the offset of a set of items within the whole GPU-allocated array, and $I_{local,i}$ is the offset of the item within the set itself. For example, for per-agent attributes, the local offset is stored in the `lidx` field of the `GpuDemand` structure (Listing 2.5), and can be used to calculate the group offset.

For millions of agents with multiple travel legs in each of the plans, and when a travel leg contains from a handful to hundreds of network links in a route, mapping itself is a computational challenge and shall be designed with maximum performance in mind. In GEMSim, a single chunk of GPU memory is allocated, partitioned between the multiple CPU binding threads, and then each of the binding threads (i) replicates the GPU data structures for a given portion of agents in host memory, and (ii) executes as few memory transfers as possible between a GPU and the host to copy the data. Typically, a GPU has multiple copy engines to transfer data between the host and the device, so a binder benefits from multi-threaded mapping processes. At the host side, each thread is mapped onto a separate logical CPU core.

In GEMSim, all objects transferred to a GPU are grouped into sets, and a whole set is always mapped to a device. Each type of these sets (that is, a population of agents, or a network of links and nodes, or a fleet of vehicles) has its own indexing implementation to better optimize the memory access of GPU threads, with different ways of sorting the items (see, for example, sorting of network links in Subsection 2.3.4). When downloading data back to the host, the same binders are used to re-map indices to IDs, and to update the host structures accordingly (e.g., for scores of individual plans, or information about traffic congestion across a network).

2.8.4 *Hardware abstraction layer*

Table 2.6 shows the hardware-specific functions that were used to implement backends for CPUs and GPUs. The implementation of the GPU backend is comparatively trivial as CUDA provides the required functions and kernels are executed in parallel, like a normal function call using the chevron syntax extension for C++. The CPU backend is a bit more complicated for two reasons: (i) atomic operations in C/C++ standard libraries do not operate directly on a memory address but rather

on special atomic data types; and (ii) a customized solution is required to run kernels efficiently in parallel on CPU cores.

The first issue can be solved by using built-in compiler functions (intrinsics), and both GCC and Clang compilers support the `__atomic_*` family of functions, while `__sync_*` or `Interlocked*` families can be used as well. The simulator build system automatically detects the presence of atomic operations and switches to one of the available implementations. Atomic operations, especially compare-and-swap (CAS), are used to avoid race conditions in the `ScheduleDemand()` kernel when multiple agents may want to depart from the same network link (hence, the same spatial buffer can be accessed simultaneously from multiple threads). The use of atomic operations, together with the probabilistic order of upstream links in the `ProcessNodes()` kernel, makes outputs of a simulation slightly different for each run.

TABLE 2.6. Hardware-specific functions used in the CPU and GPU backends of GEMSim.

Function	**CPU**	**GPU**
Memory	malloc() free() memcpy()	cudaMalloc() cudaFree() cudaMemcpy()
Atomic CAS	__atomic_compare_exchange_n()	atomicCAS()
Atomic add	__atomic_fetch_add()	atomicAdd()
Scientific	sinf(), cosf(), expf()	__sinf(), __cosf(), __expf()
Execute kernel	Pool of workers	kernel$\langle\langle ... \rangle\rangle$(...)

The second issue relates to the fact that a typical CPU, in comparison to a GPU, is less suited to very short and intensive burst loads: a context switch takes longer, and spawning a thread also takes more time. Moreover, when running on many-core CPUs, load balancing can be a problem if the workload has been statically partitioned. To overcome this issue, a pool of N worker threads was used; the algorithm of a worker thread is shown in Figure 2.35.

In it, $\mathcal{Q}$ is a queue of tasks, T_{done} is the number of already finished tasks, T_{tot} is the total number of tasks to perform, CV_Q and CV_M are condition variables, and M is a mutex. All variables are passed by pointers. The main idea here is to minimize synchronization overhead: therefore, at the beginning of the loop, workers wait until the queue receives new tasks; and at the end of the loop, only the last worker signals to the backend that all tasks are finished. The `AtomicAdd()` operation is used to count the number of finished tasks without locking the mutex twice.

The algorithm of the backend that executes a kernel in parallel is shown in Figure 2.36. At the backend side, a queue is filled with the tasks to be performed. The partitioning process splits either the network $\mathcal{G}$ (links and nodes) or the demand $\mathcal{D}$ (agents) into chunks, where each chunk represents a task for a worker thread.

```
   Input: Q, T_done, T_tot, CV_Q, CV_M, M
1  begin
2      while true do
3          LockMutex(M);
4          while IsQueueEmpty(Q) do
5              WaitOnCV(CV_Q, M);
6          end
7          T ⟵ TakeLastTask(Q);
8          UnlockMutex(M);
9          ExecuteTask(T);
10         T_done−1 ⟵ AtomicAdd(T_done, 1);
11         if T_done−1 = (T_tot − 1) then
12             SignalOnCV(CV_M);
13         end
14     end
15 end
```

FIGURE 2.35: Algorithm of the worker thread for parallel execution of GEMSim's GPU kernels on a CPU.

```
   Input: G, D, Q, T_done, T_tot, CV_Q, CV_M, M, K
1  begin
2      LockMutex(M);
3      T_done ⟵ 0;
4      Q, T_tot ⟵ PartitionTasks(G, D, K);
5      SignalOnCV(CV_Q);
6      WaitOnCV(CV_M, M);
7      UnlockMutex(M);
8  end
```

FIGURE 2.36: Algorithm of the kernel launcher for parallel execution of GEMSim's GPU kernels on a CPU.

Each task contains the start and end indices of the elements to be processed, and K specifies which kernel must be executed. After that, the backend wakes up worker threads and waits until they finish processing the tasks. Before leaving the function, the backend unlocks the mutex M as it is always locked after waiting in the condition variable. In general, a worker calls the same functions as GPU threads, but it does it in a loop for a given chunk of data. Worker threads are spawned once at the beginning of a simulation.

2.8.5 *Benchmarks*

First, the scalability of the multi-threaded data binder was evaluated using the older Switzerland scenario from Section 2.4 with 3 million cars only. Benchmarks were run on a dual-socket computing node with Intel Xeon E5-2690v4 CPUs and Nvidia P100 GPUs. Figure 2.37 shows the strong scalability for the data binding process from the host to the GPU. While for a single CPU core it takes about 15 minutes to map host data to GPU structures, 12 cores can reduce this time to less than 2 minutes. Using more than 12 cores offers only marginal performance improvement.

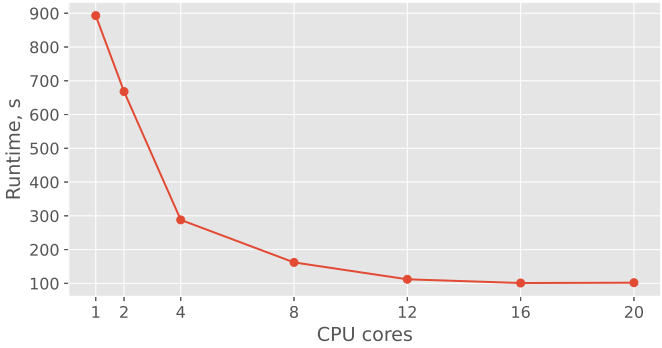


FIGURE 2.37: Strong scalability for data binding process from the host to the GPU.

Next, a large-scale agent-based scenario for Switzerland, described in Chapter 4, was used to evaluate the novel approach of using heterogeneous hardware in mobility simulations. This scenario includes 5.5 million agents (3 million cars and 2.5 million public transit users), the whole road network of Switzerland (1.1 million links and 0.5 million nodes) and the entire Swiss public transit schedule (30 000 stop facilities and 20 000 routes). An iteration covers a duration of 30 hours with 1-second resolution. Between the iterations, the learning process of the agents is performed by re-routing a random sample of 10% of the agents based on congestion patterns identified in previous iterations.

The hardware configurations used for the experiments are shown in Table 2.7. The Graviton2 configuration is the `c6g.8xlarge` instance in Amazon's EC2. For the Intel CPUs hyper-threading was switched on, while the other CPUs used only physical cores. All machines had the RedHat/CentOS (version 7 or 8) Linux distribution installed. On dual-socket systems, the `numactl` utility was used to ensure that, if possible, all simulation threads and memory banks were bound to the same physical socket. For RAM subsystems used, the number of channels varied from four to eight. Additionally, in order to quantify the improved performance by using the proposed data structures on CPUs, the same scenario was also run with MATSim on the E5 2690 configuration with 32 threads.

TABLE 2.7. Hardware configurations used to benchmark GEMSim's runtime in a heterogeneous CPU-GPU environment.

Abbreviation	CPU / cores		GPU / cores		RAM (chnls)
Gold 6150	Intel Xeon Gold 6150	2 x 36	–	–	DDR4 2667 (6)
E5 2690	Intel Xeon E5-2690v4	2 x 28	–	–	DDR4 2400 (4)
EPYC 7442	AMD EPYC 7442	2 x 64	–	–	DDR4 3200 (8)
Graviton2	Amazon Graviton2	64	–	–	DDR4 3200 (8)
K40	Intel Xeon E5-2620v2	2 x 12	K40c	2 880	DDR3 1600 (4)
P100	Intel Xeon E5-2690v4	2 x 28	P100	3 584	DDR4 2400 (4)
V100	AMD EPYC 7442	2 x 64	V100S	5 120	DDR4 3200 (8)
A100	AMD EPYC 7442	2 x 64	A100	6 912	DDR4 3200 (8)

First, using the Gold 6150 configuration, the scalability of the CPU backend, depending on the task size and the number of cores, was evaluated. For each combination of task size and the number of cores, the scenario was run 10 times for 10 iterations. The runtime was then averaged: first, averaged across the 10 iterations of each launch, and then averaged across multiple launches. As one can see in Figure 2.38, the task size impacts the runtime performance only when 10 to 20 cores are run, and the difference is up to 19% for task sizes of 4 096 and 8 192. The use of more than 20 cores yields only marginal improvements in runtime performance, and the use of 32 cores results in only a 10% faster runtime. The task size of 4 096 with 32 cores, which provided the best speed-up factor of 14.53 over a single-threaded execution, is used further in this benchmark. For a single core, the runtime is in the range of 120–125 minutes on all configurations.

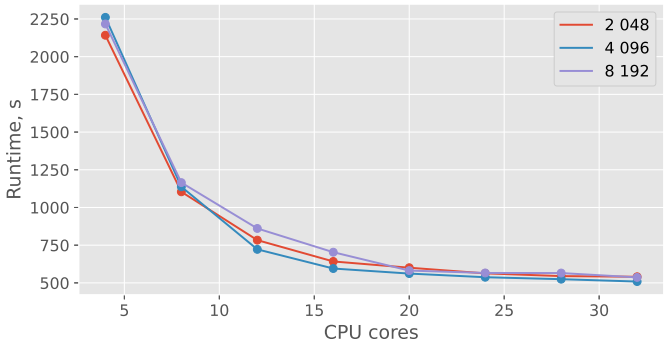


FIGURE 2.38: Strong scalability of the CPU-based GEMSim backend by the number of cores and the per-core task size.

The 10-iteration runtime averages of the CPU and GPU backends on different hardware configurations are shown in Figure 2.39. For each configuration, 32 CPU threads were used for the CPU backend and re-routing. The worst overall performance, as expected, is for the relatively old (year 2013) K40 configuration, which takes more than 15 minutes to run one daily iteration. However, it is interesting to note that almost all of the modern many-core CPUs have a similar performance for traffic propagation; this indicates that synchronization overhead and memory latency are the most important factors in improving runtime performance. Another interesting point is that each new generation of GPUs essentially doubles the runtime performance of the traffic propagation model (this is due to higher memory bandwidth, more cores, hardware-optimized atomic operations and improved warp scheduling). In contrast, the new generation CPUs, such as the AMD EPYC 7442, result in only marginal performance improvements. The ARM CPU has a comparable performance to modern x86 CPUs, and, overall, is less than 5% slower than the AMD EPYC 7442. Lastly, it can be seen by comparing the best-performing CPU and GPU configurations that the GPU backend runs up to 3.89 times faster, while the overall iteration with learning included runs up to 1.87 times faster.

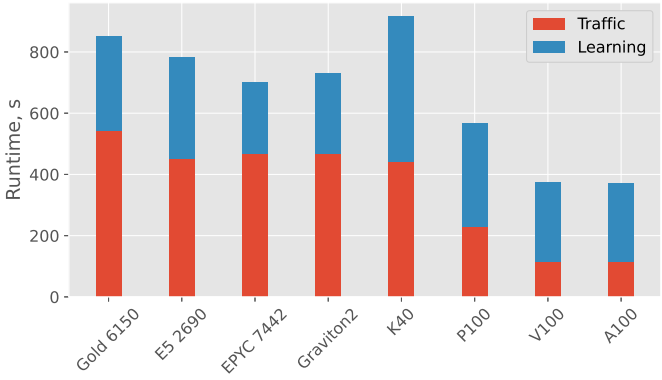


FIGURE 2.39: Performance comparison of the CPU and GPU backends for GEMSim running on different hardware configurations.

Running the same scenario with MATSim takes 87.5 minutes (single iteration only) for the traffic propagation part and 11.7 minutes for the learning part; that is, 99.2 minutes for a daily iteration. Thus, on the same hardware, the proposed CPU backend runs 11.64 times faster than MATSim, while the overall iteration is 7.62 times faster. The runtime performance of the GPU backend on V100/A100 is even more impressive: it runs 45.26 faster than MATSim, while the overall iteration is 15.84 times faster.

As one can see, only a small number of changes were required to port an existing GPU-accelerated traffic model to many-core CPUs without using OpenCL. Moreover, it is demonstrated that the same data structures can be used to achieve

high computing performances on both CPU and GPU hardware. The use of a CPU backend opens up new opportunities to further scale up agent-based mobility simulations. While GPUs have limited global memory (currently, up to 80 GB on a single board), a computing node can be equipped with much larger capacity host memory. However, in contrast to CPUs, each new generation of GPUs yields substantial improvement in runtime performance. This, however, does not hold for A100, which can be explained by the fact that, while the aggregated memory bandwidth increases from about 900 GB/s to 1 500 GB/s, the memory bandwidth is aggregated across multiple memory banks and it becomes difficult to saturate it. Another interesting outcome is that it is demonstrated that emerging high-performance CPUs, which are built upon the ARM architecture, have similar runtime performances as modern CPUs that are built upon the x86 architecture. As ARM CPUs are considered to be more power-efficient, there are opportunities for improved power consumption when large-scale traffic simulations are run.

2.8.6 *Power consumption*

Due to high demand in decarbonization and the reduction of CO_2 emissions from global society, there has been a recent trend towards green computing [212, 213], which many companies aim to support. Paralleled with the increased spread of cloud technologies for computational tasks, green computing becomes particularly important for data centres [214, 215], as well as for manufacturers and users of HPC systems [216]. One of the ways to conform to green computing ideals is to maximize the power efficiency of hardware during its lifetime, and this can be achieved not only through the development of more energy-efficient hardware, but also on the software side. In particular, certain software can be optimized for execution on more energy-efficient hardware, and today's diversity of available hardware in the cloud provides opportunities for that optimization.

In general, GPUs are more power-efficient (more operations per consumed power) than CPUs [217], hence, the use of GPUs can potentially contribute positively into the green computing initiative. As GEMSim can be run on heterogeneous hardware, producing the same output, it was possible to not only quantify a speed-up in runtime performance, but also a green-up factor F_{GR} [218], that is, the extent to which energy consumption can be reduced thanks to the use of GPUs:

$$F_{GR} = \frac{E_C}{E_{C,G}} = \frac{P_C \cdot t_C}{P_{C,G} \cdot t_{C,G}} = P_{up} \cdot S_{up} \tag{2.32}$$

where E_C and $E_{C,G}$ are energy consumptions by CPU-based and GPU-based systems, respectively; P_C and $P_{C,G}$ are power consumptions by CPU-based and GPU-based systems, respectively; t_C and $t_{C,G}$ are times to run the same simulation on CPU-based and GPU-based systems, respectively; P_{up} is the power-up factor; and S_{up} is the speed-up factor.

IPMI (Intelligent Platform Management Interface) was used to measure power consumption of computing nodes when running the same Switzerland scenario

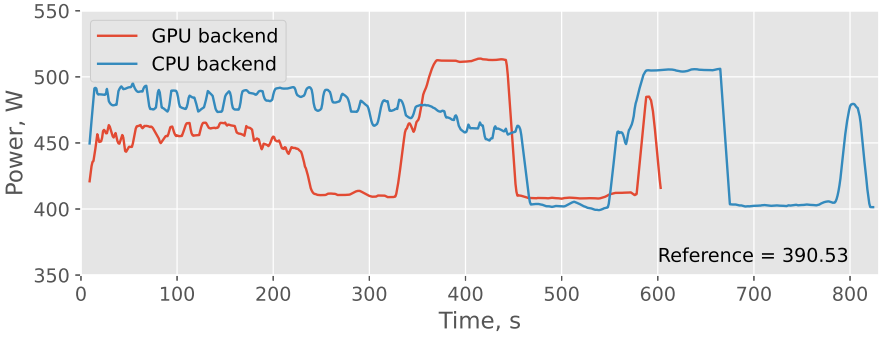
from Chapter 4 with GEMSim and MATSim. Most modern server motherboards provide IPMI to collect data from various sensors, including the instantaneous global power consumption of a node. The measurements were performed every second (wall clock time). Computing nodes were configured to run in performance mode, that is, increasing frequency of CPUs with the shortest possible delay.

Figure 2.40 shows power consumption for a single iteration (after 10 warm-up iterations) run by GEMSim with GPU and CPU backends. The reference value is the power consumption of the nodes when they are idle. Both backends provide a similar power consumption profile, which can be divided in three parts, plotted chronologically in time: (i) traffic propagation, (ii) learning, and (iii) data preparation and transfer before starting the next iteration. For both CPU and GPU backends, the peak power consumption occurs during the learning (more specifically, in the re-routing process), and it has a shape of the gate function, clearly showing the upper limits of thermal power design (TDP): around 520 W and 900 W for P100 and V100 nodes, respectively. It is interesting that the power consumption of a GPU at node V100, when running the traffic propagation part, is lower compared to the CPU backend; however, the thermal envelopes of both CPUs and GPUs have similar values of around 225–250 W. This can be explained by the fact that a GPU experiences less pressure on its computing blocks, hence it does not require throttling. On the other hand, CPUs are more sensitive to workloads, quickly raising power consumption when running multi-threaded applications.

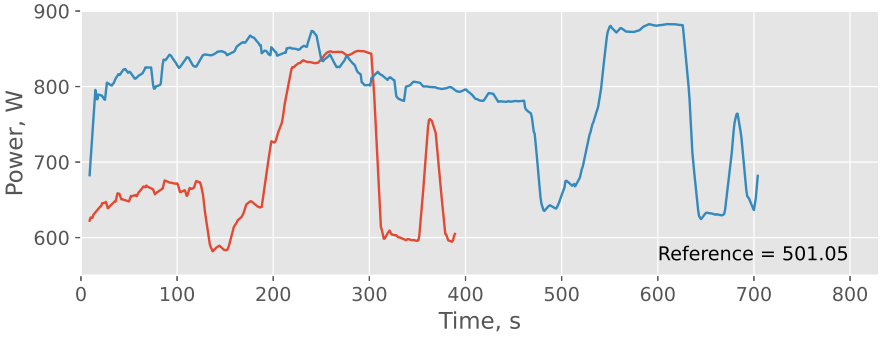
Another interesting aspect is the relative increase in power consumption of the nodes when using the CPU backend. Being an older node, P100 consumes about 120 W more (above the reference value) power during the peak load, while V100 requires almost 400 W of additional power. The most probable reason for this is that P100 has older CPUs, Intel Xeon E5-2690 v4, which have TDP of about 135 W. The inertial of returning to the idle state for V100 is also higher, probably due to longer times required to dissipate more heat. At the same time, V100 runs only about 15% faster; this could be because the traffic propagation model faces more memory and latency bottlenecks, meaning that the increase in CPU power does not directly impact the runtime performance.

The power consumption of MATSim, running one iteration of the same scenario on the V100 node, is presented in Figure 2.41. Notably, for almost the whole iteration, the power consumption remains stable above 650 W, and only during the re-routing process does it go up to 800–850 W. This, again, can be explained because MATSim does not require heavy computations to propagate traffic; rather, memory access to large and scattered data becomes a bottleneck. It could also be the reason why MATSim has a relatively poor level of strong scalability during traffic propagation, when using more than five to seven threads already brings only marginal runtime improvements.

However, one can note that when using the CPU backend and the same hardware, GEMSim has higher power consumption of around 800 W compared to the 650 W (roughly) of MATSim. As both CPU-based simulators seem to be both memory- and latency-bound, it means that GPU-optimized data structure and algorithms help to



(a) P100 computing node.



(b) V100 computing node.

FIGURE 2.40: Power consumption of computing nodes when running GEMSim with the GPU and CPU backends.

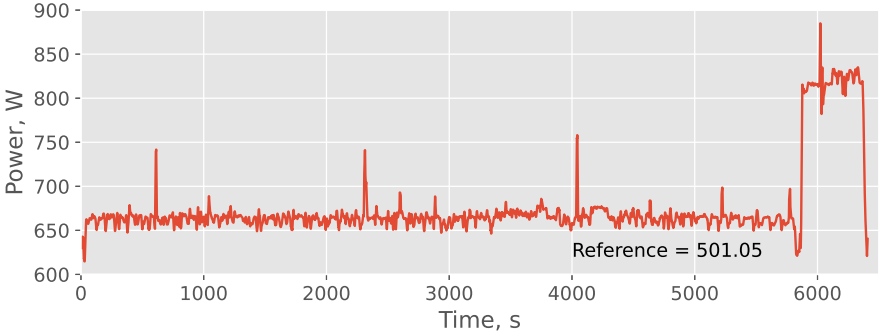


FIGURE 2.41: Power consumption of the V100 computing node when running MATSim.

utilize CPUs more efficiently, causing the cores do more work and, consequently, have higher power consumption.

Figure 2.42 shows the absolute (total) energy consumption to run the scenario for 100 iterations, which is a more practical application. The CPU-based backend of GEMSim has much higher energy consumption on both P100 and V100, with nodes in the range of 10 kWh to 15 kWh, while the GPU backend consumes about 7.5 kWh on both nodes. In contrast, MATSim requires about 140 kWh when running on the V100 node, which is more than 9 times higher than the CPU backend of GEMSim running on the same hardware, and more than 18 times higher than the GPU backend. When comparing both CPU and GPU backends of GEMSim running on the same nodes, the GPU backend provides up to 2.07 times less energy consumption.

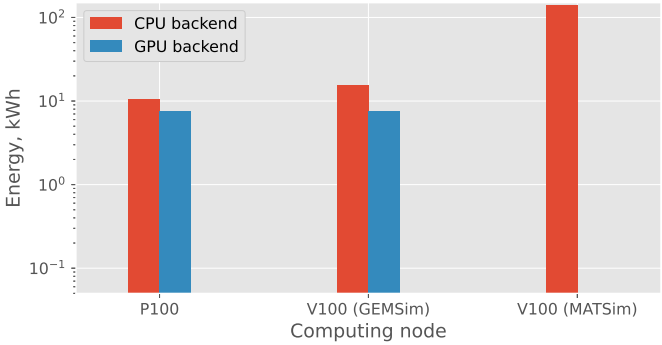


FIGURE 2.42: Absolute energy consumption required to fully run the Switzerland scenario for 100 iterations with GEMSim and MATSim.

A similar comparison of consumed energy is presented in Figure 2.43, where only the net consumption above the baseload (reference) power is reported. GEMSim consumes from about 1.5 kWh to almost 6 kWh to run a scenario with the CPU backend, and only up to about 2 kWh when using the GPU backend. In contrast, MATSim consumes about 51 kWh when running on the V100 node, which is almost 9 times higher than the CPU backend of GEMSim running on the same hardware, and more than 25 times higher than the GPU backend. The net energy efficiency of the GPU backend, compared to MATSim, is even higher, up to almost 55 times when using the older P100 node. When comparing both CPU and GPU backends of GEMSim running on the same nodes, the GPU backend provides up to 2.76 times less net energy consumption.

Another, perhaps unexpected finding from these energy consumption benchmarks is that, in terms of energy efficiency, it makes more sense to run mobility simulations on older hardware with TDP of CPUs around 120–150 W, and older GPUs which do not require equipment with higher baseload power. One can also look at these results as a trade-off between time and energy consumption per simulation. When time does not matter, energy efficient (and cheaper) hardware will be more suitable and, potentially, more cost-effective overall. However, if the simulation speed is

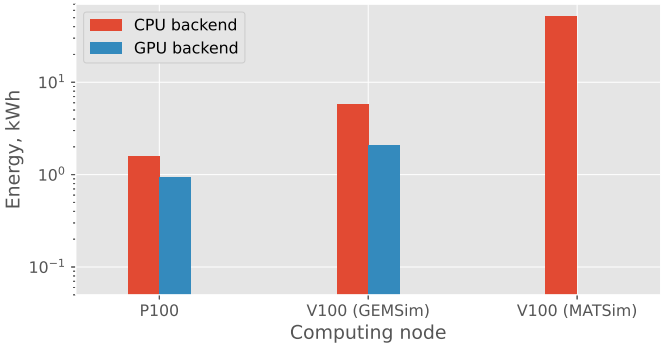


FIGURE 2.43: Net energy consumption required to fully run the Switzerland scenario for 100 iterations with GEMSim and MATSim.

of the utmost importance, then a GPU-based backend shall be used with modern GPUs, which means a higher upfront investment cost, as well as higher maintenance costs due to increased energy demand for simulations.

Green-up factors for different combinations of hardware and simulators are summarized in Table 2.8. When GEMSim runs on the same hardware, a GPU backend provides a positive contribution to energy savings, as well as running much faster. Compared to MATSim, green-up factors are even more impressive, especially when GPUs are used.

TABLE 2.8. Green-up factor (ratio of Configuration 1 to Configuration 2) for different combinations of simulators and hardware.

Configuration 1	**Configuration 2**	**Node**	**Green-up**
GEMSim, GPU	GEMSim, CPU	P100	1.68
GEMSim, GPU	GEMSim, CPU	V100	2.76
GEMSim, GPU	MATSim	V100	25.14
GEMSim, CPU	MATSim	V100	9.09

There are also two points that should be mentioned here. First, with the spread of cloud computing, many organisations are looking at how to optimize their bills, which depend on the computing materiel rented and the duration of its usage. In general, CPU-only computing instances are much (multiple times) cheaper compared to instances with accelerators like GPU or FPGA attached. Moreover, instances with accelerators have less flexibility in terms of available configurations. For example, if one needs only a single GPU but with lot of host RAM (which is very typical for traffic simulations), it may only be possible by renting a computing instance with multiple GPUs, hence stranding the resources. Here, running a

simulation longer with the CPU backend may result in overall cheaper costs for cloud infrastructure, and a user can decide whether to optimize for simulation time or the cost of running. In case consumed energy is accounted separately, that would be another dimension of the final decision.

Second, this flexibility of cost optimization is possible thanks to the support of heterogeneous hardware in GEMSim, and it shows how important it is to have such support implemented in the models to be run in a modern computing environment. Hence, for performance demanding models, heterogeneous computing should be reconsidered regarding what type of hardware to support and which frameworks to use. Such decisions are better made before starting model development, as the algorithms or data structures used can make it costly to switch later to other hardware.

3

MULTI-MODAL EXTENSIONS

> *Transport of the mails, transport of the human voice, transport of flickering pictures – in this century, as in others, our highest accomplishments still have the single aim of bringing men together.*
>
> — Antoine de Saint-Exupery

The chapter is based on contributions from the following publications:

> Saprykin, A., Chokani, N. & Abhari, R. S. GEMSim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios. *Simulation Modelling Practice and Theory* **94**, 199 (2019)
>
> Saprykin, A., Chokani, N. & Abhari, R. S. Accelerating agent-based demand-responsive transport simulations with GPUs. *Future Generation Computer Systems* **131**, 43 (2022)

This chapter presents multi-modal extensions of GEMSim which have been developed and integrated through the course of this thesis. This is one of its main contributions to the field, as other works in the area of GPU-accelerated mobility modelling are only capable of running a single mode of transport, typically cars. The ability to run multi-modal scenarios on a GPU brings more complexity into the GPU code and makes it challenging to optimize as the logic of agents becomes non-linear. Moreover, depending on the simulated transport mode, a more sophisticated interaction between the host and the GPU is required during the simulation process, including fine-grained access to the device memory.

3.1 FLYOVER

In many cases, certain transport modes may not be possible to simulate either due to unimplemented functionality, or the absence of the required input data (for example, a missing public transit schedule). However, one can estimate some average movement speeds of agents using such transport modes, and the speed to approximate a time delay when moving from one location to another. For example, walking mode can be simulated by using Manhattan distance and an average human walking speed.

GEMSim has support for so-called flyover modes, where a flyover travel leg stands for an approximate transport mode when an agent is delayed for a predefined amount of time t_{fo}:

$$t_{fo} = d_{fo} / v_{fo} \tag{3.1}$$

where d_{fo} is the flyover distance (typically, the Euclidean distance is used), and v_{fo} is the average movement speed for this flyover mode. Multiple flyover modes with different speeds can be specified for a simulation scenario. Essentially, when an agent departs from an activity location with a flyover travel leg, the demand scheduling kernel, presented in Figure 2.10, updates the scheduled time t_{sch} of the agent. Figure 3.1 presents the logic used for flyover travel legs in the `MultiModeLegHandler()` function in the demand scheduling kernel. When an agent with a flyover travel leg, detected by the value of l_{type}, has the `Delayed` state, then it means that a previously set delay has expired, and the agent can proceed to the next activity. Otherwise, the agent has just started the travel leg, so the scheduled time t_{sch} is updated with a calculated delay, and the state of the agent is set to `Delayed`. In general, flyover travel legs are modelled through a simple state machine and do not require substantial computing power or memory bandwidth from a GPU device.

```
   Input: S, D_t, agent, l_type, t_sim
1  begin
       /* Logic executed for other modes ... */
2      if l_type = Flyover then
3          state ⟵ GetAgentState(D_t, agent);
4          if state = Delayed then
5              ScheduleActivity(D_t, agent);
6          end
7          else
8              t_delay ⟵ GetFlyoverDelay(D_t, agent);
9              SetScheduledTime(D_t, agent, t_sim + t_delay);
10             SetAgentState(D_t, agent, Delayed);
11         end
12     end
13 end
```

FIGURE 3.1: Algorithm for demand scheduling of flyover travel legs on a GPU.

The structure of a flyover travel leg in GPU memory, in addition to the base leg structure, contains an estimated delay value with origin and destination network links. The structure of a flyover travel leg has fixed size and can be pre-allocated in GPU memory before a simulation starts.

3.2 PUBLIC TRANSIT

Public transit systems play important social, economic and environmental roles in many countries. First, effective public transit systems positively contribute to

sustainable development at the city level [219–221] by attracting the users of energy-intensive transport modes. Second, the development of public transit positively impacts issues related to societal needs through the increase of transport diversity, especially for the older population and people with disabilities [222]. It improves road safety [223] and helps to maintain social capital within and between communities [224, 225]. Third, public transit brings economic benefits to a society through effective land-use [226], more compact development and reduced urban sprawl [227, 228]. Fourth, public transit provides health benefits by shifting people towards transport modes involving more physical activity like cycling and walking [229–232].

While being important for dense urban areas, public transit systems are also becoming more complex and increasing in scale, providing services not only in city core areas but across large metropolitan areas with millions of inhabitants. Therefore, the modelling of public transit systems is essential for urban scenarios, making the GPU-accelerated implementation of public transit in GEMSim an important step towards the practical applicability of the simulator.

3.2.1 *Background*

Simulation of a public transit system is an essential part of many existing mobility simulators [76, 93, 100–102]. However, only a few works demonstrated multi-modal distributed mobility simulations [62, 63], while one other demonstrated CPU-based parallelized simulation of public transit [103].

3.2.2 *GPU-based implementation*

The input public transit schedule comprises two sets, H^T and R^T, which are public transit facilities and transit routes, respectively. A stop facility $H_k^T \in \mathcal{T}$ is described, with its coordinates, name, unique ID and the network link to which it is attached. Transit routes are grouped into a transit line, where each transit line represents one or more routes. While a transit line is what people get used to in reality, a transit route represents variables of the line, for example, movement in different directions, or some departures on the line that have different terminal stations. In general, from simulation perspective, transit lines do not present any computational burden, and the sole role is to provide a logical aggregation for inputs.

Each transit route $R_k^T \in \mathcal{T}$ comprises three main parts: a route profile, a network route and a list of departures. A route profile defines arrival and departure time offsets at stop facilities; offsets are relative to the departure time from the terminal stop. One can also indicate if a vehicle waits until the departure time specified in the schedule, or if it can leave as soon as possible. A network route simply specifies the sequence of links that a driver will follow to complete a transit route. A list of departures contains times of departure from the terminal stop, where a specific vehicle can be linked to each of the departures. Each transit route and departure

has a unique ID, similar to stop facilities. A fleet of public transit vehicles can be provided as an optional input to GEMSim, otherwise it will be automatically generated using some default specifications.

The modelling of public transit on a GPU poses some additional challenges compared to cars. First, the logic of agents becomes more complex. For example, while a car driver can finish an activity and depart the place, a transit passenger has to wait until a vehicle for a specific transit route arrives. Moreover, a passenger needs to board and leave the vehicle, requiring them to track the vehicle along the route. In CPU-based models, especially event-driven ones, this behaviour can be modelled through the subscription method, when an agent can subscribe for certain events of other agents, and the logic of the subscribed agent is implemented in the respective event handlers. When an event happens, subscribed agents are notified, and the event handlers are automatically executed. As the main idea of the GPU-accelerated mobility model was to eliminate event-driven systems, the subscription method could not be applied to model public transit.

Another limitation when using CPU-based approaches to model public transit on a GPU is the lack of on-device dynamic memory allocation. One can use queues, one per stop facility, for agents waiting for vehicles to arrive; once a vehicle arrives, the driving agent can iterate over the waiting passengers and notify for boarding. The same can be implemented for each vehicle, when a queue of on-board passengers is checked at each stop to notify to leave. This approach to queues will require extra GPU memory to be allocated, and, in general, it represents a traditional event-based system adapted for GPUs. However, as there are uncertainties involved with the delays of public transit vehicles, it is not possible to predict the size of the queues, which is knowledge required for stop facilities, and to allocate memory on the device. One can also define the maximum queue size for stop facilities, which can be adjusted during the calibration process to fit the passenger demand; but this can cause difficulties with evacuation scenarios and when simulating large crowded public events.

Instead of using queues for public transit demand modelling, an approach with state machines was implemented. In the public transit model, agents have two different roles: a transit driver and a transit passenger (rider). A transit driver behaves as a normal agent except that when traversing the route, they check the stops and the links where the transit stops are located. A driver executes only a single departure of a single route $R_k^T \in \mathcal{T}$ over the network $\mathcal{G}$ between the terminal stations, and therefore there can be multiple drivers who follow the same route but at different times. Drivers are artificially created agents that are not part of the input travel demand. Integration of public transit into the network propagation model allows public transit vehicles to share lanes with cars.

Integration of driver behaviour into the demand scheduling was done through two additional agent states: `Parking` and `Parked`. When a driver has a transit stop along the route, the state is switched to `Parking`, and then to `Parked` when an empty parking slot is obtained. This is the first integration point of the public transit system into the GPU-accelerated framework, and this is done in the `MultiModeNodeHandler()`

function from the `ProcessNodes()` kernel (Figure 2.8). The algorithm for arriving at stop facilities by a driver of a public transit vehicle is presented in Figure 3.2. Here, the *agent* is checked for performing a transit travel leg using the role, and in the case that it is a transit driver, the current network link e_{in} is checked for the link h_{next} of the next stop facility from the transit route. In case a required stop facility is reached, the driver is switched to the `Parking` state, and the returned `True` value indicates to the `ProcessNodes()` kernel to remove the agent from the traffic propagation process. In any other case, nothing happens to the agent at this point.

```
   Input: S, D_t, e_in, agent
1  begin
       /* Logic executed for other modes ... */
2      role ⟵ GetAgentRole(D_t, agent);
3      if role = TransitDriver then
4          h_next ⟵ GetNextStopLink(S, D_t, agent);
5          if e_in = h_next then
6              SetAgentState(D_t, a, Parking);
7          end
8      end
9      return TrueIfHandled;
10 end
```

FIGURE 3.2: Algorithm for driver agents of public transit vehicles checking stop facilities along routes on a GPU.

Another integration point is in the demand scheduling kernel (Figure 2.10) when the function `MultiModeLegHandler()` is called (the same one used to schedule flyover travel legs). Here, public transit passengers depart for their rides. A public transit passenger is a normal agent with two additional states: `Waiting` and `Riding`. A transit passenger does not participate in the network propagation model, but only switches between `Waiting` and `Riding` states, and checks when to board or leave the vehicle. The part of the function responsible for the departure of the passengers is presented in Figure 3.3. The function checks if an *agent* has just finished performing an activity and the next travel leg has the public transit mode; if so, the state of the agent is switched to `Waiting`. If not, nothing happens to the agent. It should be noted that a public transit driver has travel legs of network type, exactly the same used by car drivers, so only public transit passengers have travel legs of public transit type.

The interaction between transit drivers and passengers was implemented through the parking slots. Each public transit stop $H_k^T \in \mathcal{T}$ has an array of slots where drivers can park vehicles. The parking is performed by putting the unique ID of a

```
   Input: S, D_t, agent, l_type, t_sim
1  begin
       /* Logic executed for other modes ... */
2      state ⟵ GetAgentState(D_t, agent);
3      if l_type = Transit and state = Activity then
4          SetAgentState(D_t, agent, Waiting);
5      end
6  end
```

FIGURE 3.3: Algorithm for the departure of public transit passenger agents on GPUs.

transit driver into a slot. A passenger agent checks at each t_{sim} step if a driver who runs a required transit route has arrived at the desired transit stop to board or leave the vehicle. When boarding, only a transit route ID for the driver is required, but to leave a passenger agent checks a departure ID as multiple drivers who serve the same route may park at the same stop at different times.

The main logic of the public transit driver-passenger interaction model was implemented in the `MultiModeDemandHandler()` function of the `ScheduleDemand()` kernel (see Figure 3.4). This is the third integration point of the public transit system. The function `ParkTransitVehicle()` atomically acquires a parking slot at a transit stop facility. When all the parking slots at a stop facility are occupied, the driver will attempt to park at each next simulation step until they succeed. Functions for boarding or leaving a vehicle keep track of the occupancy statistics and adjust the scheduled departure time of the driver if the process takes too long. A passenger agent uses function `TransitVehicleArrived()` to check when a vehicle arrives to the stop facility where they board or leave, depending on the current state of the agent.

When the scheduled time t_{sch} of a parked public transit vehicle, initially set by the `ScheduleDeparture()` function, and later updated by boarding and leaving passengers, comes, and the driver agent is dispatched back to the network, the parking slot is released atomically in the `MultiModeDepartHandler()` function from the demand scheduling kernel. This is the fourth and the last integration point of the public transit model.

The structure of a public transit travel leg on GPUs, used by passenger agents, in addition to the base leg structure, contains IDs for public transit route and departure, and indices of start and end stop facilities. The route ID is used to look for vehicles while waiting for boarding, and the departure ID is used to track the movement of the public transit vehicle along the route to decide at which stop to leave. Indices of start and end stop facilities are specified relative to the terminal stop station of the transit route.

```
Input: 𝒮, 𝒟_t, agent, t_sim
1  begin
2      state ⟵ GetAgentState(𝒟_t, agent);
3      role ⟵ GetAgentRole(𝒟_t, agent);
4      if role = TransitDriver and state = Parking then
5          ParkTransitVehicle(𝒮, 𝒟_t, agent);
6          ScheduleDeparture(𝒮, 𝒟_t, agent);
7          SetAgentState(𝒟_t, agent, Parked);
8      end
9      else if role = TransitRider and state = Waiting then
10         if TransitVehicleArrived(𝒮, 𝒟_t, agent) then
11             BoardTransitVehicle(𝒮, 𝒟_t, agent);
12             SetAgentState(𝒟_t, agent, Riding);
13         end
14     end
15     else if role = TransitRider and state = Riding then
16         if TransitVehicleArrived(𝒮, 𝒟_t, agent) then
17             LeaveTransitVehicle(𝒮, 𝒟_t, agent);
18             ScheduleActivity(𝒟_t, agent);
19         end
20     end
21     return TrueIfHandled;
22 end
```

FIGURE 3.4: Algorithm for the interaction model of public transit driver and passenger agents on GPUs.

The structure of a public transit schedule on GPUs is presented in Listing 3.1. No information about transit lines is stored, as they are used only for logical grouping. The list of departures (field `deps`) contains information used by drivers, and the drivers are sorted within the local index in the same order as their respective departures; therefore, the driver's local index $I_{local,i}$ is used to access the departure information. The `stop_queue` array contains parking slots, with a fixed capacity per stop facility. Experiments showed that for the Switzerland scenario with the full-scale national public transit system, use of 10 parking slots per stop facility does not cause the vehicle to be unable to park. Increasing this number, however, does not require a lot of additional GPU memory as only four bytes per parking slot is used. The array `veh_idx` stores indices of vehicles used for each of the departures. Finally, the values of `deps_count` and `stop_count` hold the number of departures and stop facilities, respectively.

```
struct GpuTransitSchedule {
    GpuTransitDeparture deps       []; // Departures
    int32_t             stop_queue[]; // Parking slots
    int32_t             veh_idx    []; // Vehicles
    int32_t             deps_count;   // Departure count
    int32_t             stop_count;   // Stop facility count
};
```

LISTING 3.1: Structure of a public transit schedule $\mathcal{T}$ on GPUs used by passenger and driver agents for interaction.

The structure of a public transit departure is presented in Listing 3.2. It contains four arrays with the data per stop facility from the route: network links to which facilities are attached (field `stop_links`), indices of stop facilities to visit (field `stops_idx`), departure offsets (field `stops_offset`), and occupancy counters (field `occupancy_cnt`). In addition, the number of stops is stored in the `stop_count` field. While a transit driver is moving along the route, it tracks visited stop facilities by incrementing the `stop_next` field. Route and departure IDs used to interact with the passengers at parking slots are stored in the fields `route_id` and `departure_id`, respectively. Finally, the departure from the terminal stop, used to schedule departure times from next stops, is in the `departure_time`.

```
struct GpuTransitDeparture {
    int32_t stop_links    []; // Links of transit stops
    int32_t stops_idx     []; // Transit stops
    int32_t stops_offset  []; // Departure offsets, s
    int32_t occupancy_cnt[]; // Occupancy counters, per stop
    int32_t stop_count;      // Transit stop count
    int32_t stop_next;       // Next transit stop shift
    int32_t departure_time;  // Transit departure times, s
    int32_t route_id;        // Transit route ID
    int32_t departure_id;    // Transit departure ID
};
```

LISTING 3.2: Structure of a public transit departure of a transit route on GPUs.

As one can see, the structures of the public transit model combine both SoA and AoS approaches. For example, as driver agents need most of the departure data at the same time, it makes sense to use an AoS approach here to pack more data into a single memory transaction. There is a chance, though relatively small, that partially coalesced access may happen when GPU threads access vehicle data. Access to parking slots at each stop is also not coalesced, but as many departures share the same stop facilities, a single array has been used. In general, the public transit model does not provide high utilization of the memory bandwidth due to highly non-linear logic and interaction patterns for both driver and passenger agents.

3.2.3 *CPU-based implementation*

While the CPU backend also works with public transit, a straightforward port of the code from GPUs to CPUs incurs high performance penalties. The reason is because of how the interaction between drivers and passengers is organized. As both parking and departing actions are performed in the demand scheduling kernel, drivers can compete for the same resources, specifically, iterating through parking slots. However, at the same time, passenger agents at each simulation step iterate through parking slots to check for arrived vehicles. As hundreds of thousands of agents travel with public transit at the same time, this imposes a high computational burden. The main reason is that a CPU has very few cores compared to a GPU, iterative loops cause short and intensive burst loads, and a compiler may not necessarily can optimize loops properly. While a GPU can hide latencies by running many more threads, this is not the case for CPUs.

For the CPU backend, GEMSim uses a custom event notification system for passenger agents. This system extends the GPU-based data structure, `GpuTransitSchedule`, with two additional queues: one for passengers waiting to board a vehicle, and another for passengers waiting for a vehicle to arrive at a destination stop. Each of the queues contains a list of waiting agents for each of the stop facility; essentially, each stop facility has a key-value associative array where a key is the agent's unique ID and the value is the unique ID of a public transit route or departure, depending on the queue (to board or to leave a vehicle). When a public transit vehicle arrives at a stop facility, it checks respective queues, schedules agents for the next simulation step, and removes them from the queues. At the next simulation step, re-scheduled agents will start checking parking slots for arrived vehicles. Hence, passenger agents are active (checking parking slots) only when a needed vehicle arrives at the awaited stop facility.

3.2.4 *Scalability*

The scalability of both GEMSim and MATSim was evaluated and compared by using population samples of 100 000 up to 3 million agents from the older Switzerland scenario introduced in Chapter 2, but replacing 30% of each sample with public transit users. Figure 3.5 shows that GEMSim delivers significant improvements in runtime for multi-modal large-scale scenarios. Compared to MATSim, GEMSim provides a higher real-time ratio more than order of magnitude greater. Interestingly, MATSim runs faster with fewer cars simulated; for example, it achieved a ratio of 39 for a population size of 1.5 million, compared to the ratio of about 20 achieved in the car-only scenario. In contrast, GEMSim ratios are lower compared to car-only cases of the same population sizes, for example 1 060 versus 1 720 for a population size of 1.5 million. This behaviour is expected as the public transit model is less optimal for GPU execution, compared to the car-only one. Nevertheless, the improvements are

significant and, with the increasing population sample, MATSim shows a steeper drop in performance than GEMSim.

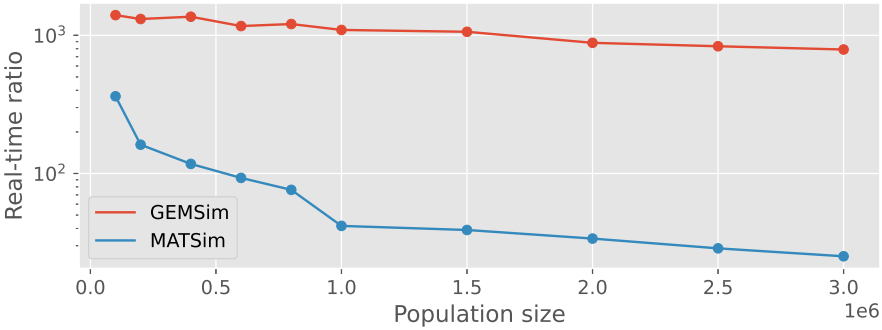


FIGURE 3.5: Real-time ratio (simulated seconds in each second of real time) of the traffic propagation part with public transit for GEMSim and MATSim.

The speed-up of GEMSim over MATSim is shown in Figure 3.6, and for the sample sizes of 1 million or more the speed-up factor is between 27 and 31. This is lower compared to the case when only cars have been modelled and a speed-up factor of over 100 was reached. The saturation point of GEMSim also shifts from 2 million agents to 1 million, due to the fact that the more complex and non-linear logic of public transit agents makes it more difficult to hide the latencies of GPU threads. The speed-up factor continues to increase at a lower rate, however, with samples above 2 million.

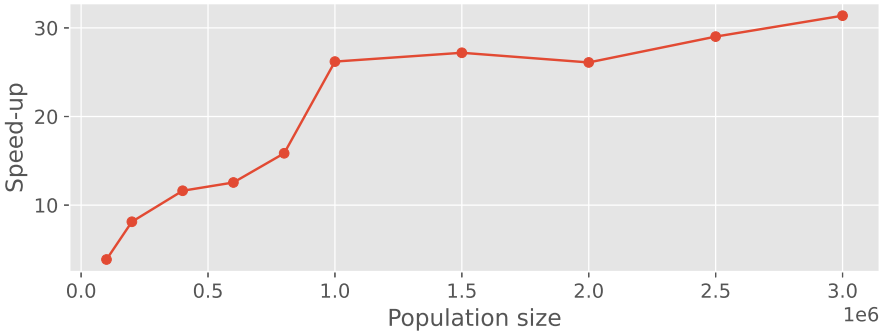


FIGURE 3.6: Speed-up factor of GEMSim over MATSim for the traffic propagation part with public transit depending on the population sample size.

Figure 3.7 shows the host RAM consumption, and like the car-only case, GEMSim consumes about five to six times less RAM depending on the population sample size. Both simulators consume about 8%–12% more memory when public transit is being

modelled, which is expected as additional input data is used, as well as routing services that build additional data structures and caches. As before, MATSim has a steeper curve for RAM consumption with the increase of the sample size.

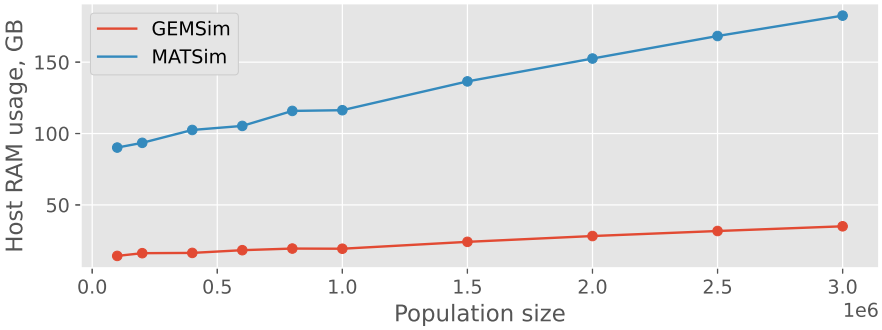


FIGURE 3.7: Peak host RAM consumption during the simulation for GEMSim and MATSim with public transit depending on the population sample size.

Figure 3.8 shows the GPU DRAM consumption for the different population sample sizes. Compared to the car-only case, the GPU memory consumption is slightly higher, which can be explained by the fact that daily-activity plans for public transit users are typically larger in terms of the number of travel legs and additional transfer activities. This difference, however, varies depending on what kind of public transit user travel profile prevails in the samples.

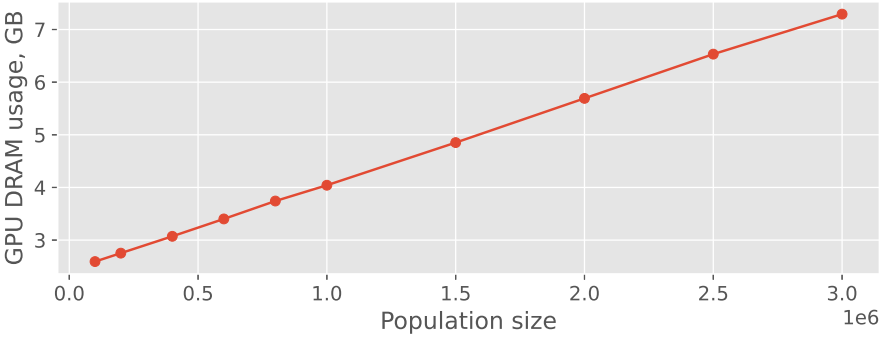


FIGURE 3.8: Peak GPU DRAM consumption by GEMSim during the simulation with public transit depending on the population sample size.

Overall, GEMSim provides a substantial increase in runtime performance (and with a smaller RAM footprint at the same time) over MATSim even with a large-scale public transit system modelled in addition to car traffic. Although the efficiency

is reduced compared to the car-only case, the improvement in runtime and RAM footprint over MATSim continues to increase with larger population sample sizes.

3.3 COORDINATED FLEETS

Demand-responsive transport (DRT), which is a more general term for taxi-like services operating upon a request (as opposed to private car and public transit modes), represents a different way of transport scheduling, as an agent may not know in advance the route that shall be taken while travelling from one location to another. The concept of DRT can be extended further into the operation of coordinated fleets, when an operator has a full control over the vehicles in the fleet. DRT becomes a very attractive in combination with emerging technologies like AVs and BEVs, as a fleet can be optimized as a whole, as well as co-optimized with energy infrastructure. Currently, there is an increasing interest from industry and policymakers in automated driving technologies and fleet coordination that can be used in DRT services to lower the operating costs and provide social benefits.

In recent years, automated driving technologies have been under very intense development, with many automotive companies targeting the production and delivery of highly automated vehicles within the next 5 to 10 years. This expected widespread availability of AVs opens up new opportunities for providers of mobility services though centrally coordinated fleets. For example, in cities and their surroundings, if reliable and cost-effective taxi-like services with fully automated vehicles are widely available, car owners may be motivated to switch to such AV taxi services [233, 234]. Furthermore, there are a number of challenges that may be addressed by the deployment of AV taxi fleets. A first example is the first and last mile problem [235, 236], that arises as people tend to own a car because public transit services are not located in walking proximity to a person's dwelling or location of work. A second example is the ever-increasing demand for parking; the substitution of private cars by AV fleets may alleviate the demand for parking and allow for the more efficient use or alternate re-use of existing parking infrastructure [237]. A third example is increased revenues for public transit authorities, which may result when AV fleets are coordinated with public transit services [238] in order to attract more users of public transit.

Notwithstanding the aforementioned potential benefits, an open question concerning the impact of AV fleets on congestion remains. While studies indicate that fewer AVs are required to serve the same travel demand of private car owners [109, 239, 240], the total travelled distance of the vehicles increases due to pickup trips and induced demand [29, 241], which may adversely affect congestion [105]. Additionally, these studies suggest different impacts on the replacement rate (how many private vehicles each centrally operated AV replaces) and the quality of service (how long a passenger waits after requesting an AV). Despite these questions and challenges, cities such as Singapore have already begun to use AV fleets for taxi service, and it is expected that in the near future more cities will follow this path.

Hence, an improved understanding of how AV fleets will be used is of importance for policymakers and city planners. Among the tools to improve this understanding are agent-based mobility simulators, like TRANSIMS [53], MATSim [76], SimMobility [242] and TaxiSim [243], that have been used to assess scenarios with AV or taxi fleets. Agent-based modelling is well suited to simulate taxi fleets: each taxi vehicle can be modelled as an agent controlled by a central operator who dispatches available vehicles in response to requests from passengers.

While agent-based simulations of taxi services using fleets have been conducted, it is evident from the review of recent literature (see below) that there is a gap to be filled by high-performance, agent-based simulation tools that are suited for large-scale, integrated deployments of coordinated fleets. In the absence of such a capability, many prior works have reduced the complexity of the models by either not simulating non-taxi traffic or by reducing the population sample. As the authors of the reviewed works have commented, high resolution, large extent simulations of both the road network and mobility services would yield simulation results that are both more realistic as well as more reliable. For example, if the modelled taxis were electric vehicles, then it would be difficult, if not impossible, to quantify the impact of battery charging on a local distribution grid if a low resolution (aggregated) simulation framework were applied. Another limitation of many prior works is that multi-modal mobility was not considered. The simulations omitted non-taxi modes of transport; however, many indications are that multi-modal trips (including coordinated fleets) are likely to be one of the most widely adopted solutions by future mobility services [244]. In order to fill this gap, GEMSim incorporates the ability to model coordinated fleets for large-scale DRT services. While this thesis is focused on taxi services, the same model can also be used for product delivery modelling.

3.3.1 *Background*

Recent works focusing on the application of large-scale taxi fleets emphasize the poor performance of existing simulation tools. The city-wide replacement of private cars in Berlin (Germany) with an optimized taxi fleet was assessed using MATSim by Bischoff and Maciejewski [104]. Different-sized taxi fleets, over the range of 50 000 to 250 000 vehicles, were evaluated to replace the 2.5 million city trips made by 1.1 million private cars. While the computing performance was not clearly indicated, it appears that it took around 3 hours for one daily iteration (typically, somewhere in the range of dozens to hundreds of iterations are required to converge to an equilibrium state) for a fleet of 100 000 vehicles. In another study of large-scale fleet deployment in Berlin, Maciejewski and Bischoff [105] assessed congestion effects for different replacement ratios of private cars. In total, up to 11 000 AVs in a fleet were used to serve the demand of more than 278 000 trips. The simulations were reported to be computationally demanding, with 51 iterations run to converge the mixed traffic, where one iteration took between 12 to 38 minutes, and the

computation with 100% AVs was the slowest. Thus, more than 30 hours of runtime was required for the simulations. Levin et al. [106] presented a general framework to model automated fleets (with and without dynamic ride-sharing) with realistic flow models. The framework was applied to Austin, Texas (USA). A relatively small network consisting of 1 247 links and 546 intersections was used together with the demand provided by 62 836 trips over 2 hours in the morning peak. The authors noted that only a sub-network of the region with 1.2 million trips was used due to performance limitations. Hörl et al. [107] studied fleet operating policies using a scenario for the city of Zurich (Switzerland), with 137 000 agents performing 363 503 trips, and up to 18 000 AVs serving the demand. Even the baseline scenario, without any fleet services, took 35 minutes to run a single daily iteration, and up to 4 hours (depending on the fleet re-balancing policy) with fleet services included.

One way to deal with the increased runtimes of mobility simulations with large-scale fleets is to use a sample of the population and to scale down the infrastructure accordingly (that is, use a fraction of the actual road capacity). In the Berlin scenario [105], the population sample was only 10% of the actual population, and the capacity of the road network was downscaled accordingly; it is evident that the demonstrated approach is impractical for simulations that include the complete population. Fagnant et al. [239] used MATSim to study the operations of an AV fleet in Austin, Texas (USA). The simulations included a realistic road network of the area with a spatial extent of 12 x 24 miles; the AV trips were generated based on household travel diary data from a survey in Seattle, Washington (USA). The MATSim framework was used to generate link-specific travel times for private cars during the day in the Austin area; only 5% of the 4.5 million trips in the area were simulated, and the capacity of the road network was correspondingly scaled due to the computational and memory limitations of MATSim. Interestingly, the authors emphasized that such a small population sample led to a loss of model fidelity.

While small population samples can significantly improve simulation runtimes, there is evidence that such small samples lead to uncertainties in the outcomes. These uncertainties are especially critical when the simulated infrastructure cannot be downscaled easily (e.g., a public transit schedule, a fleet or charging stations). Erath et al. [111] found that public transit in a Singapore scenario with a 10% population sample has artefacts of overcrowded buses; a 25% population sample improves the accuracy of the simulation. Bösch et al. [245] indicate that population samples of less than 5%–10% should not be used for scenarios with shared cars as such small samples lead to demand-supply imbalance: the reduced number of shared vehicles prevents situations of over-supply, and at the same time, leads to under-supply in the area due to reduced availability. In general, studies about uncertainties of agent-based mobility simulations with population samples [114, 115] show that at least a 25%–30% population sample is required to keep disaggregated results statistically similar to results with the full population sample.

Another way to improve runtimes is to use simplified models. Bösch et al. [240] performed large-scale simulations of AV taxis in the region of Zurich. The simulations included about 1.3 million agents with a total of 3.6 million trips that were

simulated with different sizes of AV fleets. While the authors presented and emphasized the need to model the demand in detail, a simplified version of the mobility model was instead used. In this simplified version, the travel demand and travel times were calculated in advance based on previous simulations, and the actual road network was not modelled. Instead, the movements of agents were simulated as a time delay based on a beeline distance between locations and adjusted with a multiplication factor and the average speed in the area. Thereby, no private car traffic was simulated, and the computing performance of the model was not clearly reported (but appears to be in the range of minutes to hours). In the study of Austin [239], the link-specific travel times were used in simulations to substitute the traffic from private cars when simulating the operation of AV fleets. Levin et al. [106] demonstrated that, when a realistic traffic flow model and travel patterns of a population are used, the replacement rate of private cars by AV fleets is less than in studies that use simplified traffic models; that is, one AV can substitute at most 3.6 personal vehicles compared to 9 or more vehicles found in studies that use simplified traffic models [104, 239, 240]. Hörl et al. [107] modelled fleet services in the city of Zurich without congestion and used free-speed travel times instead. The authors emphasized the need for more realistic studies to understand the impacts of congestion on fleet operating policies.

While in recent years GPU-accelerated traffic models have been developed, they remain relatively limited in functionality for simulating coordinated fleets. This section aims to demonstrate how fleet simulations were integrated with GPU-accelerated traffic models.

3.3.2 *DRT-enabled loop*

As many types of DRT services are similar (that is, a request from a client followed by the assignment of a vehicle from an operator), an initial decision was made to separate the implementation into two parts. The first part provides a generic framework for DRT services, and the second part provides an implementation of a specific type of service, that is, conventional taxis or shared taxis. This approach allows for the extension of the simulator with more DRT services in the future. This approach is conceptually close to that implemented in MATSim [246], however later development stages showed that such a separation for a GPU-accelerated model does not bring much benefit. The main reason is that GPUs are not well-suited for abstracted object-oriented code, causing code duplication and more non-linear logic of the agents, hence imposing performance penalties. Therefore, the final implementation relies on a universal model that can be configured to the scenario needs.

The DRT framework introduces the notion of an operator that provides mobility services upon a request from passengers. An operator comprises (i) a fleet of vehicles, (ii) a placer that defines the initial locations of the vehicles, (iii) a data parser that transforms the raw binary data coming from a GPU, (iv) a fleet tracker

that monitors the locations of the vehicles, (v) a fleet scheduler that assigns available vehicles to incoming requests from passengers and prepares a schedule, and (vi) a schedule dispatcher that uploads a schedule onto a GPU, notifying passenger agents about the assigned vehicles.

As an input, a set of fleet operators is described in the scenario configuration file and multiple operators can be simulated in a single scenario. The description of an operator contains a unique operator ID, name, specifications of the vehicles, and algorithms for initial placement and scheduling, operating area and some others. Instead of providing detailed specifications of vehicles, one can simply specify the number of vehicles in the fleet, and GEMSim will create an artificial fleet with a default vehicle model.

The integration of the DRT framework into the previously developed GPU-accelerated simulation loop, shown in Figure 2.6, required some structural changes; the modified simulation loop is shown in Figure 3.9. Here, the additional input of the fleet operator structure, $\mathcal{F} \in \mathcal{S}$, was required. Compared to the original loop, the `ScheduleDemand()` part now also handles DRT passengers, while two additional steps, `SyncEvents()` and `OptimizeFleet()`, were added to allow executing fleet management operations.

The decision was made to implement the fleet scheduling part completely on CPUs. The main reason is that a fleet operator has to perform extensive routing to match requests with available vehicles, as well as to provide routes from pickup to drop-off locations. As was demonstrated in Subsection 2.2.3, GPUs are not well suited for such optimization problems in the transport domain. Moreover, fleet scheduling algorithms involve highly non-linear logic, spatial filtering and other types of tasks which it is mostly not practical to run on GPUs. Therefore, in contrast to other transport modes supported in GEMSim, DRT services are very different as this mode requires not only efficient synchronization between the host and the GPU, but also a high-performance CPU-based part for fleet scheduling.

These two new steps in the simulation loop provide the points of synchronization between the traffic propagation part on a GPU and the fleet optimization part on the host side (using CPUs). An operator constantly runs the optimization loop shown in Figure 3.10. The optimization loop runs completely on the host side, communicates with a GPU at certain frequencies f_{sync} and f_{opt}, and is split into two threads which work in a pipeline manner to overlap data processing on the host with computations on a GPU. When multiple fleet operators are simulated, each of them runs its own loop in separate threads, and the synchronization points act as muxers and demuxers of data streams of different operators, meaning that each operator gets only relevant data.

The first thread loads data from a device and converts this raw binary data into the operator's input data. Two types of input data, or events, come from a GPU: (i) notices when something happens (for example, a taxi arrives at a specific location) that can be quickly updated in the fleet model, and (ii) requests for service from passengers. Binary data is stored on a GPU in a single memory chunk and

Input: $\mathcal{S}$, $\mathcal{D}_t$, t_{end}
Output: $\mathcal{S}_t^E$,$\mathcal{U}_t$

begin
  for $t_{sim} \longleftarrow 0$ **to** t_{end} **do**
    `ProcessLinks(`$\mathcal{G}$, $\mathcal{D}_t$, t_{sim}`);`
    `ProcessNodes(`$\mathcal{S}$, $\mathcal{D}_t$, t_{sim}`);`
    `ScheduleDemand(`$\mathcal{S}$, $\mathcal{D}_t$, t_{sim}`);`
    `UpdateExternalities(`$\mathcal{S}$, $\mathcal{D}_t$, t_{sim}`);`
    if $t_{sim} \mod \left\lceil 1/f_{sync} \right\rceil = 0$ **then**
      `SyncEvents(`$\mathcal{F}$`);`
    end
    if $t_{sim} \mod \left\lceil 1/f_{opt} \right\rceil = 0$ **then**
      `OptimizeFleet(`$\mathcal{F}$, $\mathcal{D}$`);`
    end
    $t_{sim} \longleftarrow t_{sim} + 1$;
  end
  $\mathcal{U}_t \longleftarrow$ `ScorePlans(`$\mathcal{D}_t$`);`
  $\mathcal{S}_t^E \longleftarrow$ `CollectExternalities(`$\mathcal{S}$`);`
  return $\mathcal{S}_t^E$,$\mathcal{U}_t$;
end

FIGURE 3.9: Algorithm for the modified GPU-accelerated simulation loop of GEMSim with integrated modelling of DRT services.

is atomically updated by GPU-run threads. Data is downloaded by `SyncEvents()` from a GPU at a frequency of f_{sync} that is currently 1 Hz.

The second thread distributes the operator's input among processing blocks: a tracker receives the locations of vehicles, and a scheduler prepares requests for assignment. At a frequency of f_{opt}, the `OptimizeFleet()` procedure flushes all data from the first thread and runs a scheduling procedure that assigns available vehicles to the collected requests and prepares a dispatch schedule. The dispatch schedule is then uploaded to a GPU by a dispatcher. The frequency f_{opt}, on one hand allows for the accumulation of requests, which results in a more optimal assignment for a set of requests or vehicles at the same time; and on the other hand improves the overall performance by reducing the number of optimizations to be performed. Currently, f_{opt} is 1/30 Hz.

3.3.3 *GPU-based implementation*

In general, the GPU-based concept of DRT services is close to public transit, except that taxi drivers do not follow fixed routes, but are rather managed by an operator.

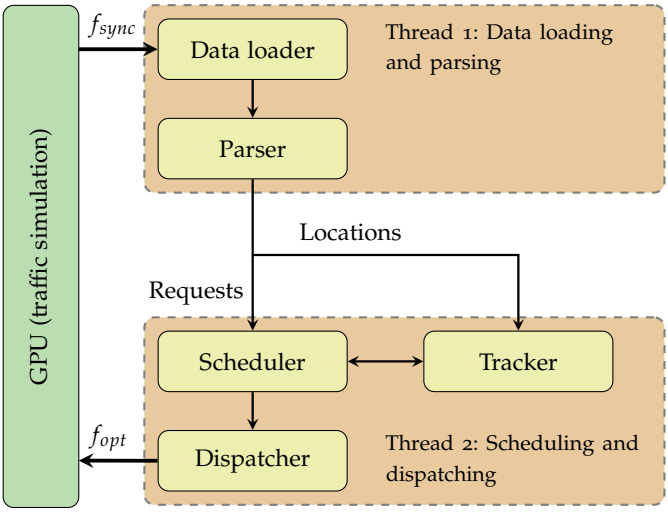


FIGURE 3.10: Schematic of the optimization loop run by a fleet operator within the DRT framework in GEMSim.

Instead of having parking slots, drivers perform pickup and drop-off activities of fixed durations, as with any other activities like work or shopping performed by the agents, and use network travel legs as any other car driving agents do. The difference, however, is that a driver has to wait (or cancel waiting after a certain time limit) until the passengers requesting the trip have boarded. It could be also that an agent pre-books a trip in advance, hence a driver may need to wait when a passenger arrives. Integration of the taxi driver's pickup behaviour was done in the `MultiModeDemandHandler()` function in the `ScheduleDemand()` kernel (Figure 2.10), and the algorithm is presented in Figure 3.11.

A taxi driver, after arriving at the pickup location, sets the number of passengers to wait there, taking this number from the user data field, `udata`, of the last travelled leg. The driver continues waiting by adjusting the scheduled time t_{sch} for one more minute each time until the expected number of passengers arrive. Waiting conditions are set up in the `MultiModeNodeHandler()` function in the `ProcessNodes()` kernel, and for a DRT driver this function also returns a `False` statement to let the kernel schedule the agent's activity in a normal way. The algorithm of the driver's actions upon arrival at a pickup location is shown in Figure 3.12.

Another integration point of the DRT services is in the demand scheduling kernel, when the function `MultiModeLegHandler()` is called. Here, DRT passengers send their requests to fleet operators. This approach is similar to that used with public transit passengers, when an agent uses two states, `Waiting` and `Riding`, to distinguish when to wait for pickup and when for drop-off. The algorithm is presented in Figure 3.13, albeit in slightly simplified form because in GEMSim

```
   Input: 𝒮, 𝒟_t, agent, t_sim
1  begin
2      state ⟵ GetAgentState(𝒟_t, agent);
3      role ⟵ GetAgentRole(𝒟_t, agent);
4      if role = DrtDriver and state = Activity then
5          t_sch ⟵ GetScheduledTime(𝒟_t, agent);
6          N_pax ⟵ GetPaxToWait(𝒮, 𝒟_t, agent);
7          if t_sch ≤ t_sim and N_pax ≠ 0 then
8              SetScheduledTime(𝒟_t, agent, t_sim + 60);
9          end
10     end
11     return TrueIfHandled;
12 end
```

FIGURE 3.11: Algorithm for the pickup behaviour of a taxi driver agent on GPUs when performing a DRT request.

agents can pre-book their DRT trips by sending requests in advance, that is, before leaving home in the morning. But without loss of generality, one can assume that agents send requests upon arrival to a pickup location.

Finally, the last integration point is in the `MultiModeDemandHandler()` function in the demand scheduling kernel. It provides the interaction model for drivers and passengers at pickup and drop-off locations, and the algorithm is presented in Figure 3.14. Here, as with public transit, agents interact through the states and passenger agents change states when boarding and leaving taxi vehicles. One important action that a passenger agent performs when boarding, in the `BoardDrt()` function, is to decrement the number of passengers for whom the driver is waiting. Atomic operations are used to update the value of this variable as multiple agents at the same simulation step may try to board the vehicle.

For vehicles from a coordinated fleet, an artificial population of driver agents is created, and the drivers use exactly the same data structure for travel legs as normal car drivers. The data structure of a DRT travel leg, used by passenger agents, in addition to the base leg structure, contains origin and destination network links, the local index of the assigned taxi driver agent (see Subsection 2.8.3 for more information), the desired operator ID to serve the request, and the current state of the request. Origin and destination links are related to the pickup and drop-off locations, respectively. The driver's local index is used to access per-driver data in the fleet operator data structure. This local index is negative until an operator assigns a specific vehicle. Link indices are used to access network properties, as well as traffic queuing buffers. The operator ID is used to indicate which service operator a passenger agent wants to reach with a request, and the state of the

```
   Input: S, D_t, e_in, agent
1  begin
       /* Logic executed for other modes ... */
2      role ⟵ GetAgentRole(D_t, agent);
3      if role = DrtDriver then
4          e_next ⟵ GetNextLink(D_t, agent);
5          if e_next = ∅ then
6              SetPaxToWait(S, D_t, agent);
7          end
8      end
9      return False;
10 end
```

FIGURE 3.12: Algorithm for behaviour when a taxi driver agent on GPUs arrives at a pickup location.

request (pending or sent) is kept to avoid duplication of the requests in the next simulation steps. It is also important to note that the size of the DRT travel leg structure is always fixed.

The structure used to store DRT operators' data is shown in Listing 3.3. The `events` buffer is used to pass the previously mentioned requests and notices from taxis simulated on a GPU to the host side, and is further described below. The array `last_link_idx` stores the last known location of each taxi and is used by passenger agents to check when to board and leave a vehicle. The array `pax_to_wait` contains the number of passengers to wait at the next pickup location for each of the drivers. The array `veh_idx` stores unique vehicle indices of drivers and these indices can be used to retrieve extended vehicle specifications (that is, a vehicle model, mass and maximum number of passengers). Lastly, the `driver_count` field stores the total number of vehicles. A single structure is used to hold data for all fleet operators from a scenario.

In order to reduce memory fragmentation and scattered access by warps to travel legs, GEMSim allocates memory on a GPU for all daily plans in a single chunk in advance and partitions this chunk for daily plans successively. While this can be done for DRT passenger legs as the structure has fixed size, this approach is not valid for driver agents as they have varying sizes of travel legs: it is not known in advance which routes will be assigned and for which fleet vehicles, so dynamic memory allocation is required. Driver agents are marked as dynamic, for which memory on a GPU is allocated individually. Moreover, the GPU typically aligns memory by the 256 bytes boundary, which means that the structure always occupies GPU memory space in multiples of 256 bytes. While the GPU can combine multiple threads when accessing sequential passenger travel legs (with some drop in efficiency because of

Input: $\mathcal{S}$, $\mathcal{D}_t$, agent, l_{type}, t_{sim}
begin
  `/* Logic executed for other modes ... */`
  $state \longleftarrow$ `GetAgentState(`$\mathcal{D}_t$, *agent*`);`
  if $l_{type} = Drt$ **and** $state = Activity$ **then**
    `SendDrtRequest(`$\mathcal{S}$, $\mathcal{D}_t$, *agent*`);`
    `SetAgentState(`$\mathcal{D}_t$, *agent*, *Waiting*`);`
  end
end

FIGURE 3.13: Algorithm for demand scheduling of taxi passenger agents on GPUs when starting a DRT travel leg.

```
struct GpuDrtOperators {
    GpuEventStorage events;          // Events

    // Per-driver data
    int32_t         last_link_idx[]; // Last link
    int32_t         pax_to_wait  []; // Number of passengers to wait
    int32_t         veh_idx      []; // Vehicle index
    int32_t         driver_count;    // Number of drivers
};
```

LISTING 3.3: Structure of DRT operators on GPUs used to store fleet vehicles, interact with passenger agents and keep data flow synced with the host.

the gaps), it is almost impossible to do the same for driver agents as the gaps are typically wider than a memory transaction size (up to 128 bytes). Information for each of the allocated memory chunks is stored in the memory registry on the host side, and only in the cases when an agent needs a larger chunk of memory does the dispatcher reallocate the memory.

In a simple case, when taxi vehicles are not shared, each assigned driver agent gets a plan with two travel legs: a pickup leg from the vehicle's last position to the pickup location, and a trip leg from the pickup location to the final destination. At the pickup and drop-off locations, a driver agent performs the corresponding activities by waiting for a passenger to board or leave (60 seconds by default, although this can be adjusted as a part of scenario input), following which the taxi becomes available for the next task. In a more general case, however, GEMSim supports multiple pickup and drop-off legs, making it possible to simulate DRT services with shared vehicles.

```
   Input: 𝒮, 𝒟_t, agent, t_sim
 1 begin
 2     state ⟵ GetAgentState(𝒟_t, agent);
 3     role ⟵ GetAgentRole(𝒟_t, agent);
 4     drv_id ⟵ GetAssignedDrv(𝒟_t, agent);
 5     pickup_link ⟵ GetPickupLink(𝒟_t, agent);
 6     drop_link ⟵ GetDropLink(𝒟_t, agent);
 7     if role = DrtRider and state = Waiting then
 8         if drv_id != -1 then
 9             drv_link ⟵ GetDrvLink(𝒮, drv_id);
10             if drv_link = pickup_link then
11                 BoardkDrt(𝒮, 𝒟_t, agent);
12                 SetAgentState(𝒟_t, agent, Riding);
13             end
14         end
15     end
16     else if role = DrtRider and state = Riding then
17         drv_link ⟵ GetDrvLink(𝒮, drv_id);
18         if drv_link = drop_link then
19             LeaveDrt(𝒮, 𝒟_t, agent);
20             ScheduleActivity(𝒟, agent);
21         end
22     end
23     return TrueIfHandled;
24 end
```

FIGURE 3.14: Algorithm for the interaction model of DRT driver and passenger agents at pickup and drop-off locations on GPUs.

3.3.4 *Synchronization of GPU and CPU parts*

Initially, the GPU-based simulation loop did not require data synchronization between the GPU and CPU parts as the GPU part with traffic propagation is always fully synchronized across all GPU threads after each simulation step (1 second). In contrast, the operator's optimization loop on the host side runs concurrently with the GPU part, meaning that data synchronization between the GPU and CPU parts was required. The events synchronization mechanism between both parts is another addition to the model which makes it possible to run fleet optimization in a GPU-accelerated mobility model.

Data transfers between CPUs and GPUs are amongst the most common bottlenecks of runtime performance in GPU-accelerated simulations. Reducing the

```
   Input: storage, event
1  begin
2  |  ev_size ⟵ sizeof(event);
3  |  cur_pos ⟵ atomicAdd(storage.offset, ev_size);
4  |  storage_event ⟵ typecast(event, cur_pos);
5  |  storage_event ⟵ event;
6  end
```

FIGURE 3.15: Algorithm of a lock-free procedure to write events into a GPU buffer.

number of memory transactions, as well as the amount of transferred data, are the most effective strategies to improve the performance. The fleet optimization frequency, f_{opt}, is used not only to make the fleet scheduling itself more efficient, but also to reduce the number of synchronized memory transactions between the CPU and GPU: many small memory transactions can be either grouped or submitted to GPU asynchronously as the GPU memory occupied by agents does not overlap.

The main data structure, used to synchronize events, `GpuEventStorage`, contains a pointer to the memory buffer that is allocated on a GPU in advance based on knowledge of how many agents are in the simulation and the maximum number of events that an agent can emit within a simulation step. In addition, an offset pointer to the current write position in the buffer and the total capacity of the allocated memory in the buffer, are stored. When `SyncEvents()` is executed in the main simulation loop, it downloads accumulated bytes of raw data from a GPU for further processing in the fleet optimization loop.

A lock-free algorithm used by GPU threads to concurrently write events into the storage is shown in Figure 3.15. First, a GPU thread reserves space in the buffer by using the atomic operation `addAtomic()` to shift the offset of the current write position: the size of the event structure is atomically added to the offset and the previous value of the offset is returned. This function, provided by CUDA SDK, has hardware acceleration in the more recent GPUs. When space is reserved, the pointer to the current write position is type-casted to the event structure, and then event data is copied into the buffer.

There are two main event structures used for CPU-GPU synchronization. The first structure is emitted by driver agents when they change their location, specified by a link ID, such that a fleet operator can track the fleet vehicles in real-time. The second structure is emitted by a passenger whenever they want to make a trip from one location, specified by the network link ID, to another location. The rest of the data in these structures are common: the type of event, ID of the agent who has emitted the event, and the simulation time when the event occurs.

In the GPU-to-CPU direction of data transfers, only events are written. When an agent wants to request a taxi, the DRT request event is written into the events buffer,

and the state of the agent is switched to `Waiting`. Similarly, when a taxi driver wants to report the location, the respective event is written into the events buffer.

In the CPU-to-GPU data transfer direction, daily plans for drivers are written and passenger plans are updated by a schedule dispatcher. For a driver agent, the schedule dispatcher first uploads the daily plan of the assigned trip reallocating GPU dynamic memory if necessary. Then, in the `GpuDemand` structure (Listing 2.5), the fields `first` and `cur` are updated for the assigned plan, and the fields `state` and `sched` are updated to switch a driver from idle to the active state. For a passenger, the schedule dispatcher updates only the local index of the assigned driver in their current travel leg structure.

3.3.5 *Memory optimizations*

Five approaches to GPU memory optimization were used in the DRT model to improve the overall runtime performance. First, coalesced memory access of GPU threads is crucial for almost any GPU-accelerated model. As demonstrated, only limited coalesced memory access is possible, mainly for per-agent attributes (state, role or scheduled update time). Passenger data structures provide limited coalesced access as the agents are active at different times of the day, and drivers are assigned to passengers in a wholly unpredictable manner. Thus, a GPU can rarely combine multiple memory transactions when passengers check if a driver has arrived from such agents. Instead, the focus in passenger data structures is to get as much per-agent data from a travel leg as possible in a single transaction. Data structures of driver agents almost exclude coalesced access, as a plan of every driver is allocated in a different region of GPU memory. This can be addressed by implementing a custom GPU memory allocator to reduce fragmentation. Second, asynchronous memory transfers were used to update the daily plans of passengers and drivers on GPUs. After fleet scheduling, a large stream of small memory transactions between the host and GPU has to be executed. Asynchronous transfers allow the GPU to aggregate and optimize a stream of memory transfers without blocking the execution of the main code. Third, it is possible to route global memory requests bypassing the L1 (on-chip) cache; this improves performance for non-uniform (that is, scattered) access patterns, and the DRT-related code was compiled with the L1 cache turned off. Fourth, some GPUs use the same hardware resources for the L1 cache and shared memory (that is, they are shared by a block of cooperatively executed GPU threads). As the code does not use shared memory, hardware resources were only allocated to the L1 cache. Finally, GPUs have fast and cached constant memory, which is as fast as a register when threads in warp access the same address. Constant memory was used to store configuration parameters and physical constants, as these variables do not change and are frequently accessed during the simulation.

3.3.6 *Fleet scheduler*

A fleet scheduling problem can be formulated as a minimization of $\sum_{i,j} W_{ij}$ where W_{ij} is the waiting time of the $i-th$ passenger if the $j-th$ vehicle from the fleet is assigned. There are other possible problem formulations, for example the minimization of empty mileage, but the idea remains the same: efficiently assign vehicles from the fleet to incoming requests based on the optimization objective.

There are two major performance bottlenecks when running large-scale agent-based scenarios that also include fleet deployment: the movement of the agents and the assignment of vehicles from the fleet to incoming requests. GPU acceleration can be used to solve the former bottleneck; however, as the fleet assignment part was implemented on the host side, GEMSim required a computationally efficient fleet-scheduling algorithm in order to keep the overall high-performance throughput. To tackle the latter bottleneck, a modified version of the scheduler based on [104, 247] was used.

First, the operating area of the fleet is split into a set of hexagonal zones using the H3 spatial indexing library from Uber [248]. This library partitions the whole globe into hierarchical hexagonal zones of 16 resolutions with an edge length from 1 000 km to less than a single meter, and provides efficient methods to map geographical coordinates to zones and back. Additionally, the library provides a convenient interface to deal with neighbouring rings of hexagons. Each hexagon from a lower resolution includes the hexagon and its six closest neighbours (called a ring) from the higher resolution level. An example of partitioning for the Zurich area is shown in Figure 3.16.

Zones are used to aggregate incoming requests and to keep track of idle taxis in separate, per-zone, registries (organized as key-value associative arrays). By default, the scheduler assumes that a taxi stays at its last position after a passenger leaves the vehicle, while there is an option available in the scheduler to assign a parking or relocation travel leg after dropping off a passenger. The assignment itself is performed as per the original version of the algorithm and is based on the idea of load-balancing supply and demand:

- During an over-supply period, the scheduler processes incoming requests in a FIFO manner minimizing the waiting time of each passenger by looking for the nearest available vehicle.

- During an over-demand period, the scheduler tries to minimize the empty mileage of the fleet by looking for the nearest request for each of the idle taxis in a FIFO manner as well.

The selection of the nearest driver or a request is performed using the hexagonal partitioning and the above-mentioned registries. The lookup area is constantly expanding from the initial zone outwards by using k-ring neighbour zones: the six closest neighbour zones first (ring-0), then the next 12 neighbour zones (ring-1) and so on. The hexagonal shape provides a good approximation of a circular area,

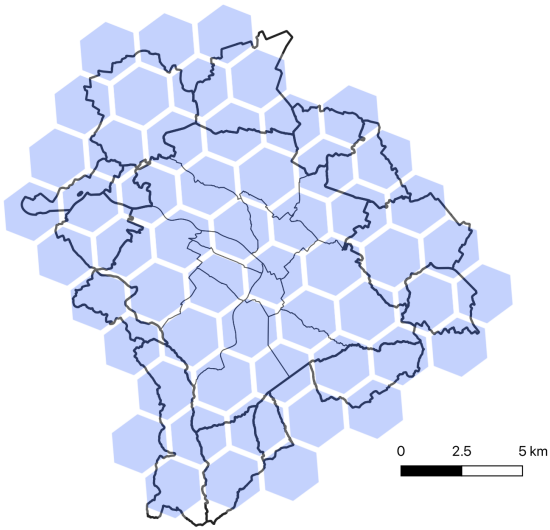


FIGURE 3.16: Example of H3 hierarchical hexagonal zone partitioning in the Zurich area used for spatial aggregation and filtering of DRT requests and vehicles.

thus one may treat found taxis or requests as the closest (by a straight line) to the central zone. The expansion is continued until at least 20 available taxis or requests are found or when all zones are checked. Afterwards, pickup and trip routes from the nearest taxi or to the nearest request are calculated using the multi-node (multiple sources or multiple sinks) A* algorithm [64] with Euclidean heuristic, an overdo factor of 2.5 and per-link congestion statistics from the previously simulated iteration. A resolution level of eight is used for the H3 library by default, and it creates hexagons with an edge size of about 470 m. While smaller zones tend to provide a more gradual expansion of the search area, they introduce (rather small) computational overhead when iterating in the lookup procedure.

Apart from the above-mentioned zone-based scheduler, other fleet schedulers were implemented in GEMSim, some based on the Euclidean distance or using global optimization algorithms as in the Hungarian method [249]. Therefore, the simulator can be used as a flexible and customizable testbed for the evaluation of algorithms related to fleet operation.

3.4 CASE STUDY: FLEET DEPLOYMENT IN ZURICH

This case study demonstrates the application of the developed GEMSim DRT modelling subsystem to evaluate potential impacts of a large-scale deployment of

an AV fleet in the city of Zurich to replace private cars. In addition, the scalability of the developed DRT model is evaluated.

3.4.1 *Scenarios*

To run all scenarios, a computing node with two AMD EPYC 7742 CPUs clocked at 2.25 GHz, 1 TB of RAM and four Nvidia V100 (32 GB DRAM on-board) GPUs was used. The older Switzerland scenario, described in Section 2.4 and in which 3.5 million agents travel by car and 1.7 million agents use public transit, was used as the basis. In the derived scenarios, a DRT operator of an on-demand non-shared taxi service was added in the Zurich area, with the following assumptions regarding fleet deployment:

- Requests are not dropped after a long waiting time, and all requests have to be served.
- While the impact of AVs on flow capacity is not fully understood, studies [250, 251] indicate that AVs can drive in a more efficient way (that is, lower headways) with an increase of up to double the flow capacity compared to conventional cars. Yet, as driving in mixed traffic may negate this improvement, AVs in the scenario use the same capacity as conventional cars and do not move faster.
- An agent posts a request for an AV immediately as soon as they want to leave their current activity place; no pre-booking is done.
- An AV is used as a personal vehicle, and only one agent can use an AV at a time.
- The initial positions of AVs are sampled from the home locations of the customers. In reality, there could be a premium paid by customers of AV taxi services to get a vehicle in the morning at a certain time, or a fleet operator will have learned where to place AVs in the morning.

While the original scenario was converged before, the switch of some agents to AVs causes a disturbance in the previously reached traffic flow equilibrium: AVs not only introduce different congestion hot spots for private cars but can also induce self-inhibited congestion [252–254] as many AVs are routed along the same streets. The reason for this behaviour is that agents with a private car know the complete congestion situation from the previous daily iteration and only a fraction of them (10%) are re-routed between the simulated daily iterations to reach the equilibrium. AVs are dispatched and routed dynamically, and all of them are routed using the same congestion statistics. In order to converge the scenario with mixed traffic, another method [105] was used. In this method, private cars are re-routed between the iterations (51 in total) in the same way as before, but AVs are routed using long-term averaged (exponential moving average with $\alpha = 0.05$) congestion statistics.

The re-routing procedure can include not only travel time but other externalities like tolls.

Even though the fleet deployment is performed in the Zurich area, the whole of Switzerland was simulated with car traffic and public transit services to keep incoming and outgoing traffic flows in the study area realistic. A full (100%) sample of the Swiss population was used, and the mesoscopic traffic model considered congestion effects for all the simulated transport modes, including AVs.

Using the basic scenario, and the above-mentioned assumptions regarding the modelling of AV fleets, two sets of AV-enabled scenarios were created. For the first set of scenarios, to evaluate a fleet deployment in the Zurich area, the AV fleet operator was limited to the city of Zurich and 14 neighbouring municipalities as many people who work in Zurich live just across the city border. Agents who use private cars in the original scenario and who travel during the day only within the defined fleet operating area are switched to AV taxi mode: 84 510 agents performing 185 770 trips out of 216 620 agents performing 515 320 trips by private car in the zone (leaving or entering), or about 36% of the trips in total, are performed by AVs. The remaining car owners outside the fleet operating area continue using private cars, and public transit users do not switch to the AV taxi service at all. The average Euclidean distance of a trip performed by agents switched to AVs is about 3.4 km or about 5 km of the travelled distance. Spatio-temporal distribution of the demand (where trips originate) for AV taxis within the defined fleet operation area is presented in Figure 3.17. For scheduling, the area was split in 319 hexagonal zones (resolution level of seven), and empty zones without any roads were removed during pre-processing.

Multiple fleet sizes of 8 500, 10 500 and 12 000 vehicles were considered according to the values from the literature. These fleets should provide replacement rates for private cars of about 10, 8 and 7, respectively. The scenario was run for 100 iterations for each of the fleet sizes. In the first 80 iterations, agents used the above-mentioned re-routing strategy, keeping up to three of the best-scored daily plans in their memory, while in the last 20 iterations agents only picked one of the memorized plans with a probability proportional to the scores of the plans. The initial placement of AVs in the morning was fixed between iterations to prevent traffic oscillations and to improve the convergence.

The second set of scenarios, to assess the scalability of GEMSim for large fleets, comprised two synthetic scenarios: (i) the city of Zurich area, used in the first set of scenarios, in which all 515 320 trips that left or entered the Zurich area were served by a fleet of 100 000 taxis; and (ii) the whole canton of Zurich with the same number of taxis and served trips. As the canton, compared to the city, is about 20 times larger in area and about three times larger in population, the two scenarios represent different trip densities and fleet vehicle spatial distributions. For the canton scenario, the car trips to be converted to taxi trips were sampled at random. For both scenarios, a set of cases varied the number of converted trips and the fleet size from 10% to 100% in increments of 10%, thus maintaining the same trips-per-taxi ratio.

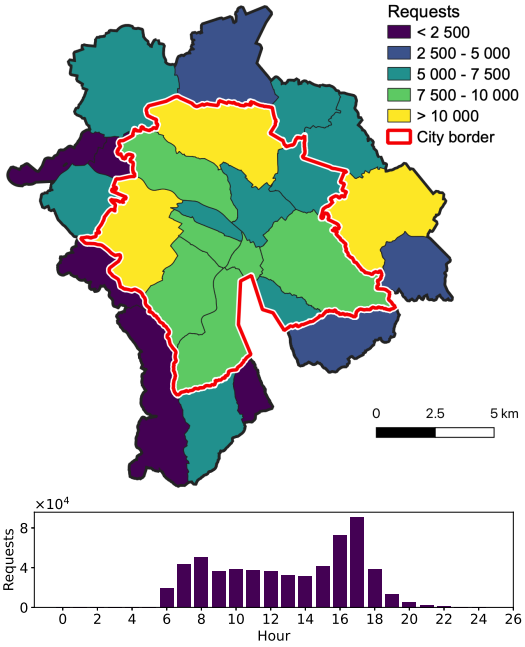
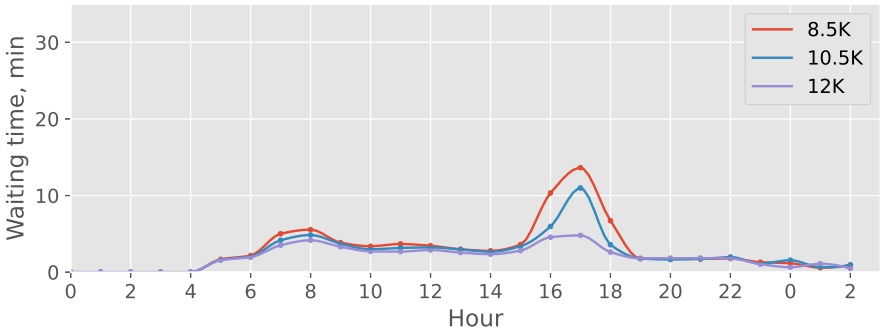


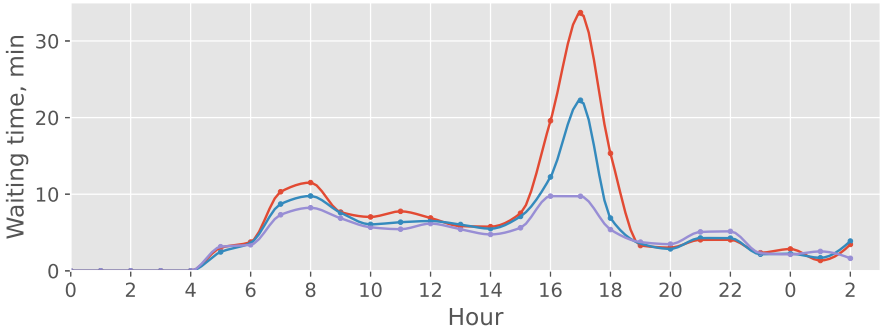
FIGURE 3.17: Spatial (upper plot) and temporal (lower plot) demand for AV taxis in the Zurich area.

3.4.2 *DRT services in Zurich*

The waiting time for passengers after requesting an AV is presented in Figure 3.18. A fleet of 8 500 vehicles can provide relatively good service throughout a day, and during the morning peak hour the average waiting time is no more than 6 minutes, and the 90^{th} percentile is about 12 minutes. However, in the afternoon peak hour when a strong increase in demand happens, the quality of service deteriorates and the average waiting time increases to almost 14 minutes, while the 90^{th} percentile goes up to 34 minutes. As expected, increasing the fleet size to 10 500 vehicles helps to reduce the waiting time during the afternoon peak hour to about 11 and 22 minutes for the average and the 90^{th} percentile, respectively. At the same time, improvement for the morning peak hour is only about a 2-minute reduction. Finally, a fleet of 12 000 vehicles can push the average waiting time in the afternoon to about 5 minutes, while the 90^{th} percentile is slightly under 10 minutes. Therefore, the waiting time of the passengers non-linearly depends on the fleet size, and for the Zurich area, a fleet of 12 000 vehicle provides short waiting times throughout the day.



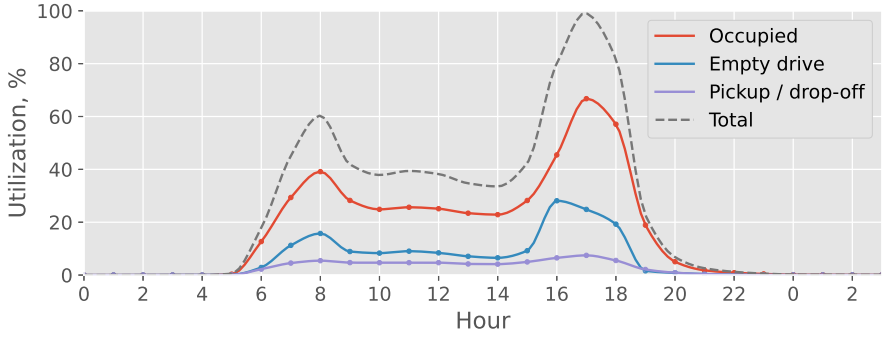
(a) Average waiting time.



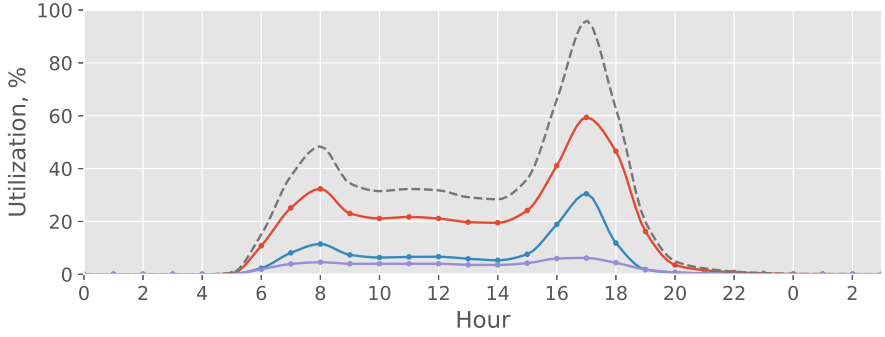
(b) 90^{th} percentile of waiting time.

FIGURE 3.18: Passenger waiting times after requesting an AV in the Zurich area for various fleet sizes.

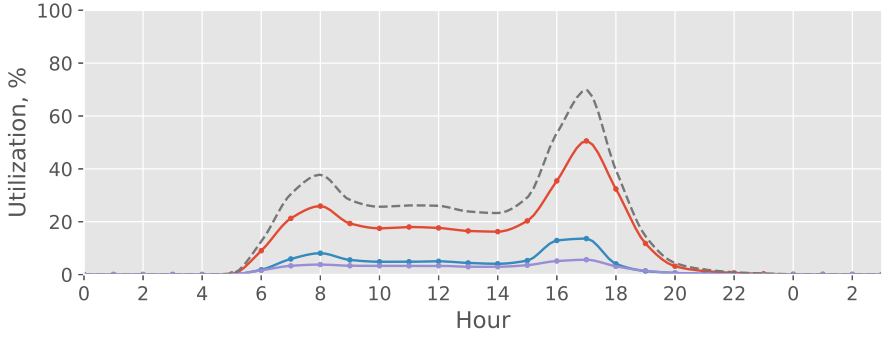
From the fleet operator's point of view, not only is the customer waiting time important, so too is the fleet utilization. This is because an under-utilized fleet is generally less profitable. Utilization of the different-sized fleets is presented in Figure 3.19. As one might expect, a larger fleet provides shorter waiting times for passengers at the cost of higher under-utilization of the fleet. A fleet of 8 500 vehicles is fully utilized during the afternoon peak hour, and it increases the probability that during the assignment process a less optimal (with a longer pickup distance) AV will be assigned to a customer. An increase in fleet size to 10 500 keeps the utilization rate close to 100% in the afternoon while providing a shorter waiting time. For both fleet sizes, empty driven time in the afternoon peak reaches 30% of the utilization, and for the fleet of 10 500 vehicles, it is half of the occupied driven time. When increasing the fleet size even more to 12 000 vehicles, the total utilization drops to 70% in the afternoon, while empty driven time contributes only about 12%. Interestingly, the time a fleet spends performing pickup and drop-off activities is quite high and contributes from 5% to 8% of the total utilization.



(a) Fleet of 8 500 vehicles.



(b) Fleet of 10 500 vehicles.



(c) Fleet of 12 000 vehicles.

FIGURE 3.19: AV fleet utilization in the Zurich area for various fleet sizes.

Another important aspect for a fleet operator is the distribution of driven distance among the vehicles, presented in Figure 3.20. As expected, vehicles from a smaller fleet drive a further distance daily: a fleet of 8 500 vehicles makes some AVs travel up to 240 km per day with rare cases of 270 km, while a fleet of 12 000 vehicles

rarely pushes AVs beyond 210 km, and less than 1% of vehicles travel this distance. For an operator, it means that a gasoline-fuelled fleet does not need to be refuelled during the day, and empty mileage to gasoline stations can be avoided. On the other hand, if an operator considers a fleet of BEVs, the distance and the battery recharging time matter. Here, a larger, under-utilized BEV fleet does not necessarily mean less profitability as for a larger fleet an operator can use cheaper EVs that have a driving range of under 300 km. Otherwise, for a smaller fleet, either a more expensive BEV model or intra-day recharging is required.

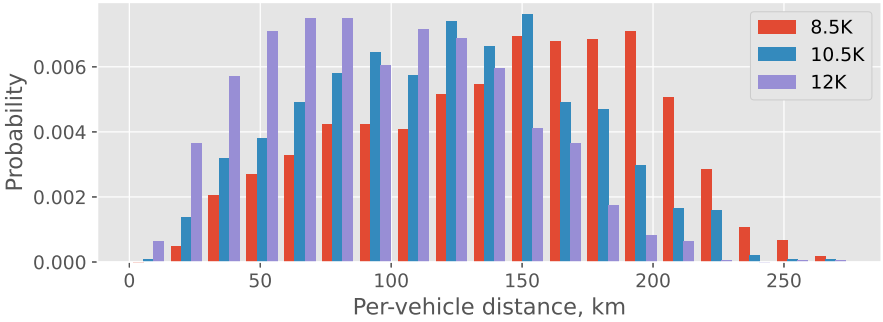


FIGURE 3.20: Distribution of per-vehicle daily travelled distance in the Zurich area for various fleet sizes.

A smaller fleet increases the per-vehicle average daily travelled distance from 95 km for a fleet of 12 000 vehicles to 144 km for a fleet of 8 500 vehicles. The total VKT is also increased with a smaller fleet size, from 1.14 million VKT to 1.23 million VKT. An increase in fleet size by 41% leads to a decrease in VKT by 8%. The share of empty mileage in total VKT decreases with the increase of fleet size and contributes from 25.1% for a fleet of 8 500 vehicles down to 19.6% for a fleet of 12 000 vehicles. As the use of an AV fleet leads to the increase of the total VKT, the average travel time of the agents within the operating area is increased from 39 to 57 minutes excluding the waiting time. Fleet size mostly does not affect this increase in travel time as empty mileage is decreased by only 5% for the largest fleet of 12 000 vehicles.

On the other side, the model also represents a trade-off between demand for parking space and demand for the flow capacity of the roads when the fleet size is changed under the constant travel demand. One of the incentives to replace private cars with a coordinated fleet is to reclaim existing parking space in a city for other needs as fewer cars in total will be on the streets. However, to keep travel times and congestion at the same levels, a city should also increase either the fleet size or flow capacities of the roads, reducing the amount of space reclaimed for other needs.

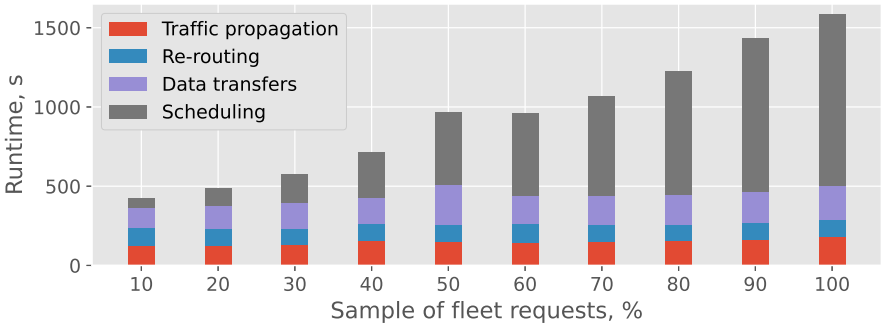
The outcomes of the case study show that the Zurich area has good potential for replacing private cars with a coordinated AV fleet. A fleet of 10 500 vehicles has a high utilization but longer waiting times, while a fleet of 12 000 vehicles has

lower utilization and shorter waiting times. This provides a replacement rate of between seven and eight private cars per AV. However, this replacement rate is lower compared to other studies performed in the same area. In one study [240], a replacement rate of 10 was achieved with a participating population of 10% and waiting times below 10 minutes. Another study of the city of Zurich [107] showed that a fleet of 7 000 to 14 000 vehicles can replace the demand of 137 000 people travelling in Zurich by car or public transit while keeping the 90^{th} percentile of waiting times below 5 minutes in the morning peak hour; however, congestion effects are not accounted for. Thus, a comparison to the present work indicates that as soon as the simulation scenario becomes more detailed and considers more realistic factors, the replacement rate of private cars by AVs decreases.

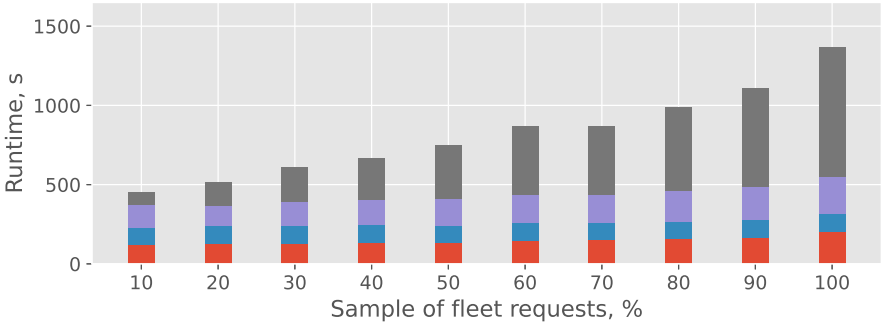
3.4.3 *Scalability*

The performance breakdown of one iteration for each of the scenarios with a large-scale fleet is shown in Figure 3.21. The first interesting finding is that the runtime for the smaller, city-scale scenario increases faster as the size of the problem increases. The reason for this is because of how the scheduling algorithm works. During times of over-supply (which is most of the day), the scheduler prepares a list of nearby driver candidates for each of the incoming requests. The greater the number of idle drivers (or, the more taxis per unit of space), then the greater the time spent preparing a list of candidates to serve a request (this is up to 20% of scheduling time for the 100% case). The second finding is that only the fleet scheduling part increases, more or less linearly, with the size of the fleet, while the other parts of an iteration remain more or less constant: only a slight increase of runtime in data transfers and traffic propagation is observed with samples. One can note that with relatively small fleets, that is with 10 000 vehicles, data transfers dominate the runtime performance of the simulated iteration. This can be explained by the fact that the scheduler is not saturated with enough work (that is, number of incoming requests and idle vehicles). However, starting with 20 000 or 30 000 vehicles (depending on the scenario), all four parts of an iteration contribute equally to the runtime performance, and only further increasing the fleet size makes the scheduler a bottleneck. This demonstrates that this approach to simulating fleets with a GPU-accelerated traffic model is effective and scalable for real-world applications.

GEMSim's runtime performance was compared with MATSim's performance. Both scenarios at full scale were run with MATSim on the same cluster node, and using the same number of CPU threads (20) to parallelize re-routing (10% of car users) of the agents between iterations. While MATSim's mobility simulator, QSim, is capable of multi-threaded traffic propagation, it was found to be limited (probably due to synchronization issues between QSim and a taxi optimizer) and only a single-threaded execution of QSim was possible when taxi fleets were simulated. The performance breakdown of one simulated daily iteration for both scenarios is compared for GEMSim and MATSim in Table 3.1. MATSim runs the scenario



(a) City of Zurich scenario.



(b) Canton of Zurich scenario.

FIGURE 3.21: Scalability of runtime performance of the GPU-accelerated DRT model using scenarios with various spatial densities of taxi trips.

with lower trip spatial density more slowly than GEMSim. This can be explained by the fact that MATSim's re-routing is about eight times slower than GEMSim's, and a scheduler has to do extensive routing on longer distances when assigning the available vehicles to requests. Overall, GEMSim is up to nine times faster than MATSim, requiring less than 30 minutes for the simulation for one full day compared to three-and-a-half hours for MATSim. The simulation time for MATSim is similar to that reported for a Berlin scenario [104] that has a similar-sized fleet but a smaller population sample.

Fleet deployment is the most computationally expensive part in GEMSim, taking 77%–81% of the simulation time; of this time, the scheduling algorithm constitutes more than 77%—83%, with the rest of the time being spent on data transfers to and from the GPU. MATSim's fleet scheduling part, while using the same algorithm as GEMSim, can run more than three times slower. This poorer performance can be explained by the fact that MATSim has a less efficient implementation of the routing algorithms used by the scheduler. Nevertheless, the overall performance improvement of GEMSim over the whole simulation loop is significant; the traffic

propagation part, which is accelerated by a GPU, contributes most of the speed-up, and runs up to 50 times faster than in MATSim.

Overall, the model for large-scale fleet deployment in GPU-accelerated mobility simulations has a substantial performance improvement compared to existing state-of-the-art solutions. However, a shift in required computing power was observed in GEMSim, and for the fleet deployment not only was GPU acceleration required, but good CPU processing power was advantageous as well. It transforms the simulator into a truly hybrid high-performance tool in which the GPU and CPU components complement each other.

TABLE 3.1. Performance breakdown of a daily iteration run by GEMSim and MATSim with a 100 000-vehicle fleet for the Zurich city area and the canton of Zurich.

Part of iteration	City scenario			Canton scenario		
	GEMSim, s	MATSim, s	Speed-up	GEMSim, s	MATSim, s	Speed-up
Traffic propagation (cars, transit, DRT)	180	9 091	50.50	205	9 202	44.89
Fleet scheduling	1085	2 109	1.94	820	2 740	3.34
Dispatching schedule to GPU	212	–	–	235	–	–
Re-routing 10% of private cars	112	890	7.95	109	898	8.24
Total	1 589	12 090	7.61	1 369	12 840	9.38

4

DEMAND GENERATION: SWITZERLAND CASE STUDY

> *All grown-ups were once children... but only few of them remember it.*
>
> — Antoine de Saint-Exupery

The chapter is based on contributions from the following publications:

> Saprykin, A., Marini, M., Chokani, N. & Abhari, R. S. *Holistic, integrated generation of daily-activity plans for Switzerland: from population synthesis to trip generation* in *20th Swiss Transport Research Conference (STRC 2020)(online)* (2020)

Over recent decades, agent-based models have been increasingly applied in many sectors, including the transport sector [44]. One of the main drivers for the increased usage of agent-based models is their ability to model complex behaviour on a disaggregated level of detail. This contrasts with four-step models that have been used in the transport sector since the 1950s. Four-step models, while being simple and computationally efficient, are unable to account for the dynamics of individual behaviour, as well as person-person interactions in transport systems. With the development of new transport modes such as car-sharing or ride hailing, these limitations are even more important. Moreover, it is a formidable challenge to apply four-step models in scenarios with Mobility-as-a-Service (MaaS) platforms. While agent-based models are well suited for these emerging challenges in mobility, the development and application of agent-based models requires even bigger datasets and more detailed information about the actors and environment to be simulated. Typically, the accuracy of an agent-based simulation directly depends on both the quality and the quantity of available data. Moreover, as the operation of transport systems also involves the complex behaviour of people, the more widespread application of agent-based models is not trivial.

As demand and supply drive any transport system, the synthetic population is a cornerstone in any agent-based transport model. Both individual attributes as well as behavioural patterns, or daily activities, must be described. Typically, the synthetic population is derived from a variety of datasets, ranging from a mobility census to statistical data of the local area to be studied. These datasets are often in incompatible formats, with different levels of resolution, and may refer to different years. They must often be pre-processed and/or converted into a customized representation in order to be analysed or visualized. Furthermore, the generation of agent-based travel demand is typically divided into the two steps: (i) synthesis of the population and (ii) generation of trip chains, which requires additional effort to make the two steps compatible. Another challenge arises when something has to be changed either in the input data or in the steps, for example in

the evaluation of scenarios, and the whole process of generating the travel demand must be repeated in a reproducible way.

Here, a unified modelling pipeline (that is, a sequence of executed models) was proposed in order to automate the generation of agent-based travel demand. The modelling pipeline, when executed, reads the required input data and transforms it into an agent-based travel demand that is ready-to-use in an agent-based mobility simulator. This approach both simplifies the maintenance of agent-based models, and allows a user to easily re-generate travel demand for specific situations, such as the spread of a viral disease (which is very closely linked to people's behaviour). Furthermore, this approach makes the use of agent-based models easier for non-experts such as policymakers, who may only want to specify a set of input parameters for a scenario that they then simulate without going into the details of the demand modelling.

4.1 BACKGROUND

The synthesis of population agents that realistically represent the actual population in the area of interest is the first requirement for the generation of travel demand and subsequent mobility simulations.

The most common approach [255] comprises two main stages: (i) population fitting, and (ii) allocation. In the first stage, fitting, a reference sample of agents, typically derived from census data, is fitted to aggregated constraints, for example the total population. The most common fitting approach is the iterative proportional fitting (IPF) procedure [256] that estimates the distribution of control variables based on the reference sample. This method is used in recent population synthesizers, such as ALBATROSS [257] and PopGen [258]. The second stage, allocation, disaggregates the fitted population amongst individual population agents and households selected from the reference sample. In some cases, the geographic placement of the households is also refined in this stage. The different methods of allocation include altered selection probability [259], a conditional Monte Carlo approach [260], or a deterministic selection [261]. Other approaches to synthesizing population agents are summarised elsewhere [262]. The main limitation of these approaches is that the quality of the resultant synthetic population depends directly on the quality of the reference sample; this sample is very often of limited size and detail for reasons of personal privacy in census data. While this part of the thesis is focused on the generation of disaggregated travel demand, it requires a synthetic population as an input. The synthetic population is generated using LEC's in-house model [263], which combines registers for dwellings and commercial activities with aggregated population registers. The population agents are synthesized through a series of models with the goal of realistically capturing the characteristics of the actual population, but without the need to use a reference population sample. Thus, this approach is more advantageous as it can be applied in many geographic locations, where the data required to generate a reference population sample are unavailable.

The synthetic population is the basis for disaggregated travel demand generation models. Activity-based models [264–266] derive travel demand from an integrated overview of people and households. These models are based on the demand for socio-economic activities, where individual agents have spatio-temporal constraints [267] and move between different locations incurring time and travel costs. Activity-based models place emphasis on the fact that participation in certain activities generates travel demand and complex interactions. Agents typically try to maximize their utility function [268] by building and scheduling a sequence of trips between the locations of activities. Many daily-activity models are tour-based, whereby an agent starts and ends their day at the same location, usually home. Rule-based algorithms [269, 270] are most widely applied in activity-based demand generation, but it is noted that these algorithms tend to approximate the process of activity scheduling [271]. However, as activity-based models are used to model the socio-economic behaviour of individuals, they are difficult to implement and calibrate; hence, many such models are applied only at a regional or local level [272, 273]. Another approach to generating travel demand for agent-based transport simulation models is to sample trip chains directly from microcensus data [274]. In this approach, trip chains are sampled from statistically similar respondents and adjusted in time and space. The main advantage of this approach is its simplicity, as few assumptions are made regarding the process of activity scheduling, and the approach can be applied at the scale of census data.

This part of the thesis integrates the latter approach to generate trip chains for synthetic agents, thus avoiding the need to develop and calibrate econometric models of people's behaviour. However, mobility microcensus data is required to apply the unified modelling pipeline described in this chapter to another geographic area. Given that many countries conduct mobility surveys on a regular basis, the unified modelling pipeline can be applied to a wide range of cases.

4.2 MODELLING PIPELINE

The structure of the modelling pipeline is shown in Figure 4.1. The pipeline is organized as a sequence of steps, which are executed in order to transform raw input data into agents' daily-activity plans for a mobility simulator. The pipeline can be split into two main parts: a population synthesis model and a daily-activity model, with the daily-activity model taking the output of the population synthesis model as an input. The whole modelling pipeline was implemented in Python, and, for reasons of improved performance, parts of the population model were accelerated with a GPU using the *Numba* framework [275]. While this chapter describes the modelling pipeline using the Switzerland case, the main structure is flexible and can be adapted for other geographic areas: see Chapter 5 for the application of the pipeline to the Munich metropolitan region.

Population synthesis, being a constituent of demand generation, is out of the scope of this thesis, and only a brief description is provided. The structure and

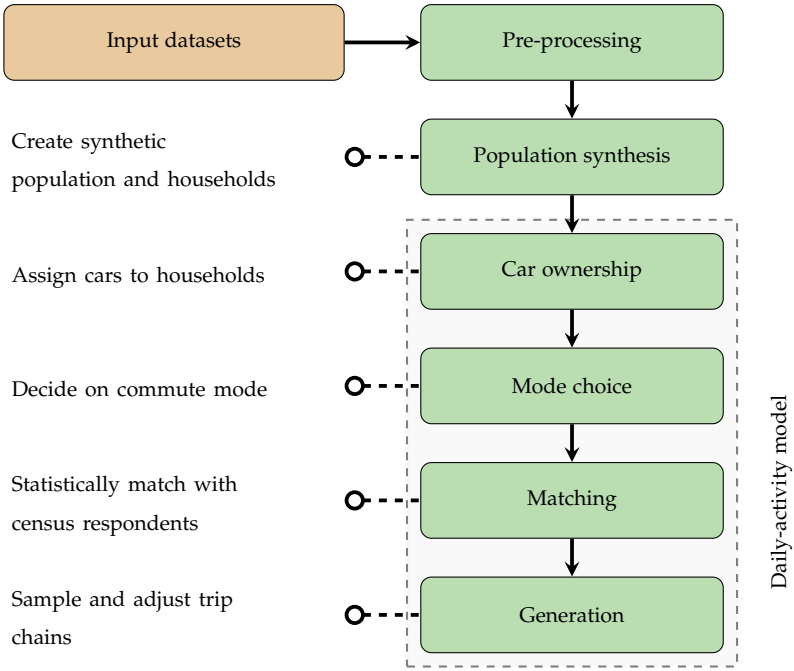


FIGURE 4.1: Structure of the agent-based travel demand modelling pipeline.

data flow of the population synthesis is presented in Figure 4.2. In the agent-based population synthesis framework, four typologies of geo-referenced agents are modelled, specifically:

- **Population agents**. These represent individual persons who perform their daily activities and travel between the locations of activities.
- **Household agents**. These represent individuals who are grouped into a family unit, and who live in the same dwelling.
- **Job agents**. These represent available and occupied jobs that are assigned to agents that have a matching skill set.
- **Dwelling agents**. These represent available and occupied residential locations, in which households live.

The generation and linking of the synthetic population for the reference year can be subdivided into the following steps:

- Individual population agents are generated, starting from demographic data that characterises population agents in terms of age, sex and education level,

and are extrapolated as a function of age from the Adult Literacy and Life Skills Survey for Switzerland [276].

- Individual job and dwelling agents are generated based on the Business and Enterprise Register [174] and the Register of Residential Buildings and Dwellings [277], while the costs of dwellings are assigned based on the data taken from online listings.
- Population agents are linked to job agents in the vicinity of the municipality as a function of the distribution of skills; the commuting matrix, described in the Mobility and Transport Microcensus (MTMC) [278], is used to define the probability of working in a specified district.
- Individual agents are linked in households, starting with the distribution of household typologies in the municipality [279]. Partners are matched assuming minimization of difference, with compatible sex and sexuality, while children are assigned to parents as a function of the age of the female partner.
- Residential dwellings are assigned to households, such that size and cost of the dwelling matches the structure and income of the household.

The generated synthetic agents are then used as an input for the daily-activity model that generates input daily plans for GEMSim.

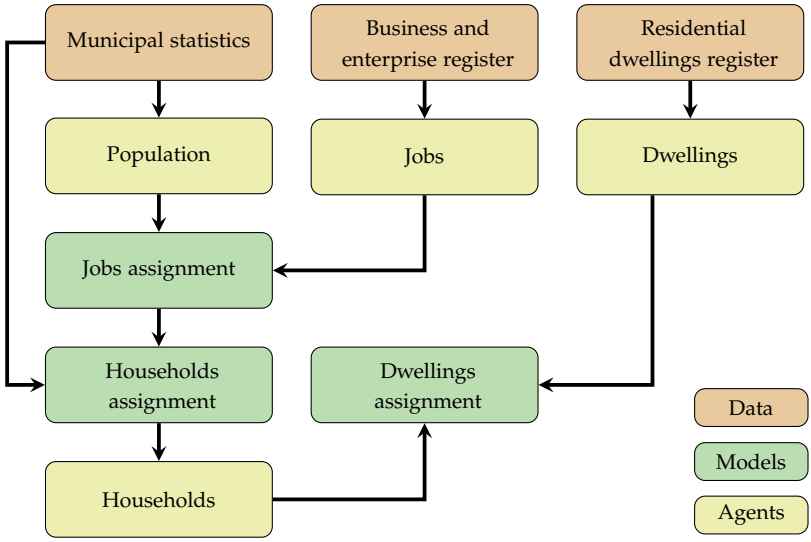


FIGURE 4.2: Structure of the agent-based population synthesis model used to provide input for demand generation.

4.3 DAILY-ACTIVITY MODEL

The structure and data flow of the daily-activity model is shown in Figure 4.3. The model incorporates other sub-models including DCMs for car ownership and transport mode; as well as input datasets together with the output from the population synthesis model, which are propagated successively until the daily-activity plans are generated.

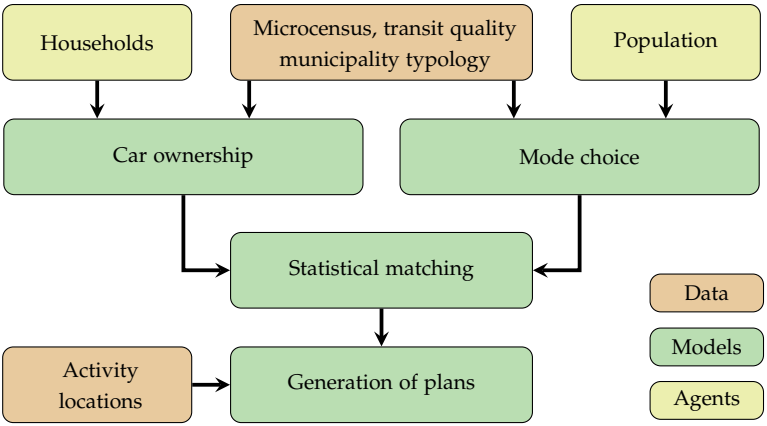


FIGURE 4.3: Structure of the agent-based daily-activity model used to generate individual travel plans.

4.3.1 *Pre-processing*

The following datasets are inputs to the daily-activity model:

- Synthetic population.
- Mobility and Transport Microcensus (MTMC) [278].
- Administrative borders [280].
- Municipality typology: urban, sub-urban, rural [281].
- Public transit quality map [282].
- Postal code boundaries [283].
- Locations of places of activity from OSM [149].
- Car register (MOFIS) [284].

The microcensus provides information about the personal attributes and the detailed travel behaviour of 57 090 participants in Switzerland. Both the synthetic population and microcensus datasets contain comparable attributes of agents as required by the daily-activity model:

- **People**: age, gender, job location.
- **Households**: location, size, typology, income.

The travel behaviour in the microcensus is described as a set of individual daily trips, 193 880 in total. In the daily-activity model the following trip properties are used: start and end locations, departure and arrival times, transport mode, travel purpose, and travel distance. The microcensus data can be filtered as either a typical working day or weekend; as a result, variations in behaviour within a week can be distinguished. Here, a typical working day is used as an example, but the developed model can also generate travel demand for a typical weekend.

Municipalities are classified into one of three categories based on the Gemeindetypologie [281] classification, which accounts for spatial attributes such as population density and accessibility. This classification was used to estimate car ownership and transport mode choice models. Figure 4.4 shows the classification of municipalities in Switzerland used in the daily-activity model, while the mapping of typology from the Gemeindetypologie classification is shown in Table 4.1.

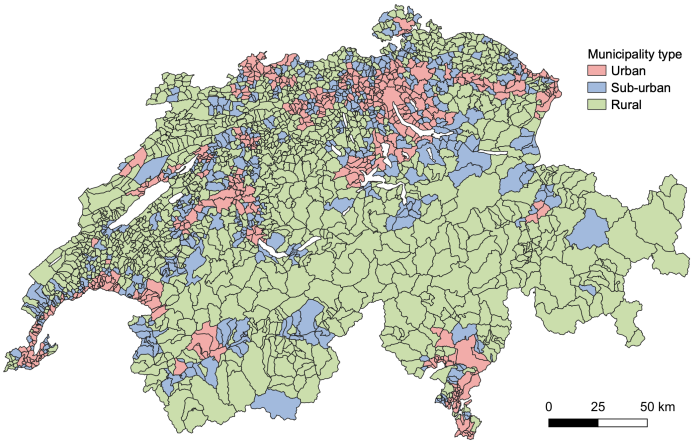


FIGURE 4.4: Swiss municipality typology based on the Gemeindetypologie classification.

The public transit quality map (ÖV-Güteklassen) specifies four classes (A, B, C and D) of public transit service quality depending on the proximity of stops, the variety of transport modes in the area and the frequency of service. The public transit quality map for the canton of Zurich is shown in Figure 4.5. Areas outside these four classes are considered to not have public transit quality. As agents travel

TABLE 4.1. Mapping of Swiss municipality typologies from the Gemeindetypologie classification.

Gemeindetypologie (code)	Typology
Municipality of a large agglomeration (11)	Urban
Municipality of a medium-sized agglomeration (12)	Urban
Municipality of a small or outside an agglomeration (13)	Sub-urban
Periurban high-density community (21)	Urban
Periurban medium-density community (22)	Sub-urban
Periurban low-density community (23)	Rural
Rural centre (good connection) community (31)	Rural
Rural centre (local) community (32)	Rural
Rural peripheral community (33)	Rural

to perform activities throughout a day, the locations of places of activity are required. These locations were taken from OSM (Geofabrik) and their types were mapped to the purpose of the trip from the microcensus. Trips with a business purpose include all available types of activities. In total, 123 577 places of activity were downloaded from OSM. The car register (MOFIS) contains detailed information about registered vehicles in Switzerland at the resolution of ZIP-code (i.e., post code); this register was used to validate the car ownership model.

After reading, the input data was cleaned. The purpose of cleaning is to remove incomplete samples that cannot be used in DCMs or which contain insufficient information to generate a proper daily-activity plan for an agent. Examples of removed samples include trips with loops (start and end in the same location), unknown transport mode, trip chains not starting or ending at the home location, or respondents without any trips reported. In total, 34 028 trips and 15 335 respondents were removed from the microcensus dataset. After cleaning, the synthetic and microcensus households data were merged with the municipalities classification and public transit quality map; thus, each household was assigned its attributes from municipality-related datasets and the quality of public transit service based on the location of the household.

4.3.2 *Car ownership*

A car ownership model was used to assign the number of cars owned by each of the synthetic households. The multinomial logit (MNL) DCM was applied to synthetic households to estimate the number of cars. The MNL model assumes that random terms are identically and independently (iid) distributed (Gumbel distribution). The

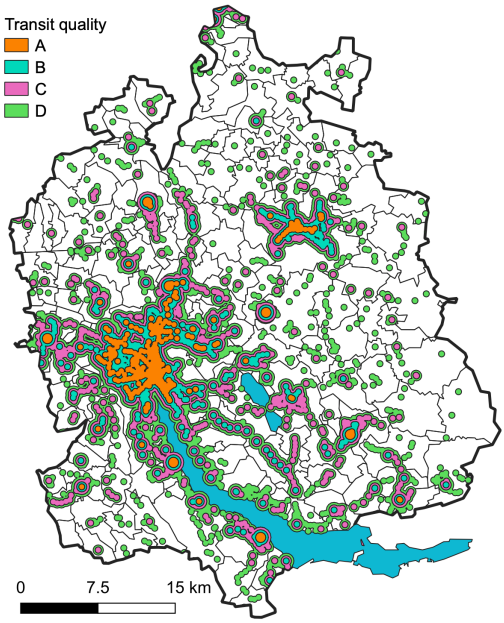


FIGURE 4.5: Map of public transit quality in the canton of Zurich, Switzerland.

probability of choice of a given alternative i for a decision-maker n is defined as follows:

$$P_{in} = \frac{e^{V_{in}}}{\sum_j e^{V_{jn}}} \tag{4.1}$$

In discrete choice modelling, the probability of each alternative is based on a set of attributes that reflect the cost and benefits of an alternative, and the utility function U is defined as follows:

$$U_{in} = V_{in} + \epsilon_{in} = \beta_i x_{in} + \epsilon_{in} \tag{4.2}$$

where V_{in} is the deterministic part of the utility function based on the vector β_i of taste parameters and the vector x_{in} of alternative attributes, and ϵ_{in} is the non-deterministic part of the utility function.

Microcensus data was used to estimate the parameters of the MNL model. The following household variables were observed to have a strong impact on the choice: monthly income (SFr, Swiss franc), typology, public transit quality and municipality typology. The variables are used as categorical, and the available alternatives are as follows: no car, one car, two cars, and three or more cars. In the microcensus, 13 712 respondents did not specify income levels and these samples were excluded from the MNL model estimate. The coefficients of the MNL model were estimated using

the maximum-likelihood method from the *statsmodels* package for Python, and the results are presented in Table 4.2. Alternatives are estimated against the alternative of not having a car.

TABLE 4.2. Results of MNL estimation for car ownership in Switzerland. Pseudo-$R^2 = 0.1379$ and Log-Likelihood is -35174.

Variable (dummy)	**1 car**	**z**	**2 cars**	**z**	**3+ cars**	**z**
Income (2 000–4 000)	0.28	2.98	0.12	0.74	0.03	0.13
Income (4 001–6 000)	0.82	8.56	0.92	5.61	1.00	3.52
Income (6 001–8 000)	1.01	10.13	1.40	8.52	1.66	5.86
Income (8 001–10 000)	1.07	9.90	1.77	10.42	2.18	7.62
Income (10 001–12 000)	1.04	9.03	1.85	10.57	2.41	8.31
Income (12 001–14 000)	1.01	7.47	1.88	9.98	2.60	8.68
Income (14 001–16 000)	0.98	6.80	1.98	10.18	3.00	9.94
Income (> SFr. 16 000)	1.06	7.80	2.22	11.83	3.50	11.82
Household (single)	0.37	3.03	-1.78	-9.63	-3.48	-11.15
Household (non-family)	0.35	2.37	0.17	0.88	-0.63	-2.05
Household (couple)	1.02	7.94	0.69	3.82	-1.03	-3.55
Household (couple+children)	1.12	8.46	1.08	5.82	0.17	0.60
Household (single+children)	0.67	4.85	0.08	0.44	-1.21	-3.95
Transit quality (A)	-1.98	-19.94	-3.47	-31.72	-4.09	-29.57
Transit quality (B)	-1.46	-14.76	-2.45	-23.19	-2.93	-23.59
Transit quality (C)	-0.93	-9.59	-1.61	-15.72	-1.90	-16.67
Transit quality (D)	-0.44	-4.45	-0.76	-7.37	-1.00	-8.91
Municipality (sub-urban)	0.19	3.79	0.36	6.47	0.52	7.48
Municipality (rural)	0.21	2.83	0.46	5.61	0.74	7.88

The results show that every variable is statistically significant. Income does not particularly impact the alternative of having a single car; however, a higher income increases the probability of having multiple cars. Households of a single person have a lower probability of having a car, while an increase in household size (including children) increases the probability of having one or two cars. Public transit quality in the area of the household's dwelling has a very strong impact on the decision to have a car: the higher the quality of public transit, the more negatively the alternatives of having a car are correlated. In general, households located in sub-urban and rural areas tend to have more than one vehicle.

To validate the car ownership model, car register data at ZIP-code resolution was used. Non-private cars were removed from the register, as well as all types of non-personal vehicles like trucks and agricultural machines. Figure 4.6 shows the relative error of the predicted number of cars at ZIP-code spatial resolution compared to the actual number of registered cars. One can note that the model captures the trend in the data: most of the high-density areas have a relative error below 10%, some areas have up to a 20% error, and very few areas (mostly close to the border) have a higher error. Most areas with high (>30%) relative errors are located in low-density mountainous regions of Switzerland, and many of these areas have in the range of a few dozen to a few hundred registered cars. Finally, one should also account for the fact that the car register gives only an approximate spatial distribution of the actual locations of registered cars. That is, for tax reasons, cars can be registered in one canton but used in another.

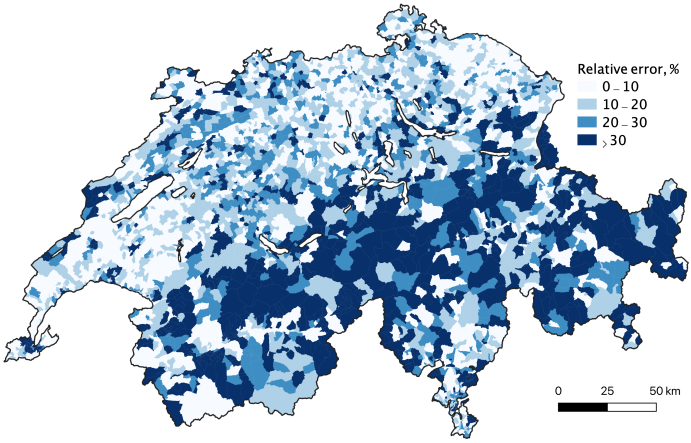


FIGURE 4.6: Relative error between predicted car ownership and car register data in Switzerland using the unified modelling pipeline.

4.3.3 *Mode choice*

Like the car ownership model, a mode choice model was implemented using MNL regression with two alternatives: car or public transit (including walking). The microcensus data was additionally filtered as follows, prior to model estimation:

- Agents below the legal driving age (18 years in Switzerland) are assigned to public transit and excluded.
- Agents living in households without cars are assigned to public transit and excluded.

- Agents who are passengers, not drivers, are excluded.

The coefficients of the MNL model are summarised in Table 4.3. The alternative of taking public transit was estimated against the alternative of using a car.

TABLE 4.3. Results of MNL estimation for transport mode choice in Switzerland. Pseudo-$R^2 = 0.1348$ and Log-Likelihood is -9518.

Variable (dummy)	**Public transit**	**z**
Age (18–24)	-0.37	-0.65
Age (25–44)	-1.85	-3.19
Age (45–64)	-1.73	-2.99
Age (>64)	-1.79	-3.10
Gender (female)	0.55	15.155
Income (2 000–4 000)	0.02	0.16
Income (4 001–6 000)	-0.03	-0.18
Income (6 001–8 000)	0.07	0.45
Income (8 001–10 000)	0.03	0.21
Income (10 001–12 000)	0.26	1.54
Income (12 001–14 000)	0.42	2.36
Income (14 001–16 000)	0.49	2.71
Income (> SFr. 16 000)	0.43	2.44
Employed	-0.55	-10.70
Transit quality (A)	1.41	17.970
Transit quality (B)	0.89	12.060
Transit quality (C)	0.67	9.785
Transit quality (D)	0.35	5.222
Cars in household (2)	-1.4392	-32.420
Cars in household (3)	-2.1771	-25.749

The results show that persons older than 25 tend to switch to a car rather than to public transit, while, in general, females prefer to use public transit. Interestingly, persons with high monthly income (more than SFr. 12 000) tend to prefer public transit. This can be explained by the fact that travel in first class is quite comfortable and/or the dwellings of these persons are centrally located. As expected, if there is a higher quality of public transit in an area then the probability of using public transit increases, while the availability of a car in a household reduces the probability of

using public transit. Finally, in general, employed persons prefer to use a car over public transit.

Figure 4.7 shows the predicted share of agents who select public transit in Swiss municipalities. In urban areas that have a high quality of public transit, people tend to drop a car, while in mountainous and low-density regions, where the quality of public transit is insufficient, the trend is opposite.

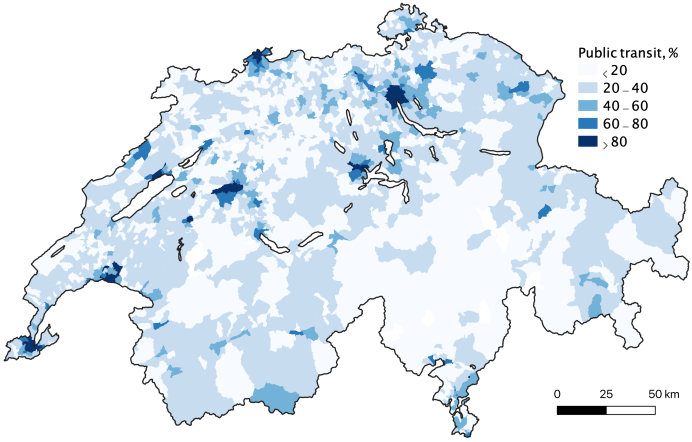


FIGURE 4.7: Predicted share of agents who take public transit for daily trips in Switzerland using the unified modelling pipeline.

4.3.4 *Generation*

The generation of the agent-based travel demand comprises three steps:

- Finding statistically the best donor of an activity chain for each synthetic agent derived from the microcensus dataset.
- Sampling the locations of activities to match the activity chains relative to the locations of homes.
- Converting the activity chains into output format and writing to a file.

All steps are performed in parallel, wherein one thread handles a block of synthetic agents, so the model is scalable as the size of population is increased. Each thread processes its own block of agents and writes output to a file. After all working threads are finished, the main thread consolidates the output files into a single output file.

In the first step, two sets of attributes for an agent in the synthetic population are compared with the same sets of attributes of each respondent in the microcensus

dataset. The first set contains mandatory attributes (of age class, gender, employment status and commute mode) which must be exactly matched for a respondent from the microcensus in order to be accepted as a donor candidate. The second set contains optional attributes (of income class, household typology, municipality typology and distance from home to work) which should be closely matched. A minimum of 30 donors is required for a synthetic agent to be accepted for generation of the plan, otherwise the agent is discarded. After comparison, the 30 donor candidates with the best overall match scores are selected, and a final donor is sampled uniformly at random from this pool. In total, all 5 542 305 synthetic agents were matched using 28 328 donors from the microcensus.

As activity chains from the microcensus contain the specific coordinates of the locations that are visited, these coordinates must be re-sampled from the OSM dataset for each agent from the synthetic population based on their household and job (if any) locations. A Monte Carlo type re-sampling of the locations of activities was made for each trip in the chain, and the chain always starts and ends at home:

- Randomly pick a direction in the range of 0–360 degrees.
- Select a point located at the trip distance from the last place of activity in the direction picked at the previous step.
- Find the closest place of activity with the corresponding trip purpose.

When locations of activities for a whole chain are re-sampled, the total travel distance between re-sampled locations is compared with the total travel distance of the donor. If the travel distance difference is less than 200 m then the re-sampled chain is accepted, otherwise the re-sampling procedure is repeated for this chain. The number of re-sampling attempts was set to 200, as this provides a good compromise between the accuracy of re-sampling and the runtime performance of the model. If after 200 iterations a solution is not found, the best re-sampled activity chain is accepted. The timeline of a re-sampled activity chain is shifted randomly within a 30-minute interval. Finally, a re-sampled activity chain is written to the output file.

4.3.5 *Adaptation for COVID-19*

To demonstrate the flexibility of the developed pipeline, it was modified to simulate the behaviour of people during the public health intervention measures imposed by the Swiss government to prevent the spread of COVID-19. Thus, an additional configuration that describes the transformation of people's behaviour during epidemics was added to the generation part of the pipeline. The transformation of behaviour can be defined in the following ways:

- Specify types of OSM locations which are closed and to which agents cannot therefore go.

- Specify NOGA codes (general classification of economic activities) for jobs that are unaffected by the measures and to which employees have to travel to work (that is, these employees do not work from home).

- For each trip purpose in the microcensus, specify a probability that an agent will abandon a trip of this purpose in their daily plan. Abandoning working activities means that an agent works from home.

- For agents who have a car in a household but do not use it, specify the probability of using a car instead of public transit.

Using the modified pipeline, four scenarios, based on data from mobility reports from Google [285], Apple (as of April 14, 2022, Apple no longer provides the data) and locally observed behaviour, were simulated:

- **No closing** (February 22, 2020): no limitations of places of activity; 0.30 probability of dropping any activity; jobs in healthcare, public services and groceries are kept running; 0.1 probability of switching to a car;

- 1^{st} **partial closure** (March 13, 2020): schools, universities and leisure facilities are closed; 0.45 probability of dropping any activity (except already closed); jobs in healthcare, public services and groceries are kept running; 0.2 probability of switching to a car;

- 2^{nd} **partial closure** (March 16, 2020): schools, universities and leisure facilities are closed; 0.70 probability of dropping any activity (except already closed); jobs in healthcare, public services and groceries are kept running; 0.3 probability of switching to a car;

- **All closed** (March 20, 2020): all facilities except explicitly allowed are closed; 0.95 probability of dropping any activity (except already closed); jobs in healthcare, public services and groceries are kept running; 0.4 probability of switching to a car.

Figure 4.8 compares the number of travelling agents throughout a day for each COVID-19 scenario. The model clearly shows that there is a change in mobility behaviour, especially in the 1^{st} partial closure scenario when educational and leisure facilities were closed. Further measures, such as stricter social distancing, do not have a strong impact on the number of people travelling daily.

Table 4.4 summarises the reduction in average travel distance and time for the scenarios. The predicted average travel distance and time were compared to real tracking data obtained in the MOBIS project [286], in which more than 3 000 participants were tracked with mobile phones and travel diaries. The real tracking data shows that after the introduction of the strictest measures on March 20, the reduction in average travel distance by car was about 50%. Thus: (i) the 2^{nd} partial closure scenario is the most realistic; and (ii) more people than expected continued to travel even after restrictions were imposed.

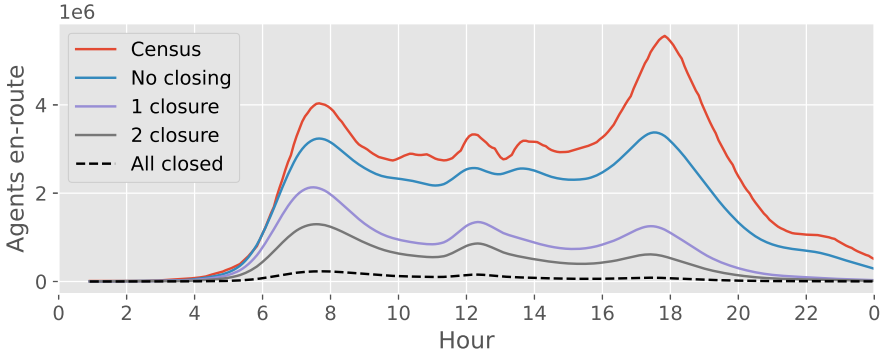


FIGURE 4.8: En-route dynamics of the agents throughout a day for COVID-19 scenarios with various governmental measures applied in Switzerland.

TABLE 4.4. Reduction in average travel distance and time for COVID-19 scenarios in Switzerland.

Scenario	Avg. dist, km	Avg. time, h	Red. dist, %	Red. time, %
Baseline	34.62	1.42	0.00	0.00
No closing	28.49	1.12	17.70	21.13
1^{st} partial closure	19.47	0.74	43.76	47.89
2^{nd} closure	17.47	0.65	49.53	54.23
All closed	15.71	0.57	54.62	59.86

4.4 BENCHMARKS

The runtime performance of the unified modelling pipeline was evaluated on a GPU computing node equipped with four Nvidia P100 GPUs, two Intel Xeon E5-2690 v4 CPUs clocked at 2.6 GHz and 256 GB of RAM. Each CPU has 14 physical cores and can run up to 28 threads in parallel when a physical CPU core represents two logical cores in the system. The runtime performance of the pipeline is shown in Figure 4.9. Depending on the number of CPU cores used, a simulation takes 4 to 5 hours and about 10 GB of host RAM to generate travel demand for Switzerland from raw input data. When using more than 20 threads the performance of the pipeline does not improve much and may even degrade. One reason for the degraded performance could be the lack of multi-threading (address space is shared among threads within a single process) in Python where multi-processing (each process has dedicated address space) is used instead. Multi-processing may lead to data segmentation and less efficient utilization of the CPU cache. Moreover, an operating system may schedule resources less efficiently when dealing with the large number of

processes (more expensive context switch). Another explanation is that the number of physical CPU cores becomes a limiting factor because a single physical CPU core has performance drop when running more than one thread in parallel. Out of total runtime, the population synthesis takes about 2.5 hours to run, while the rest of the time is spent to generate daily-activity plans of the agents.

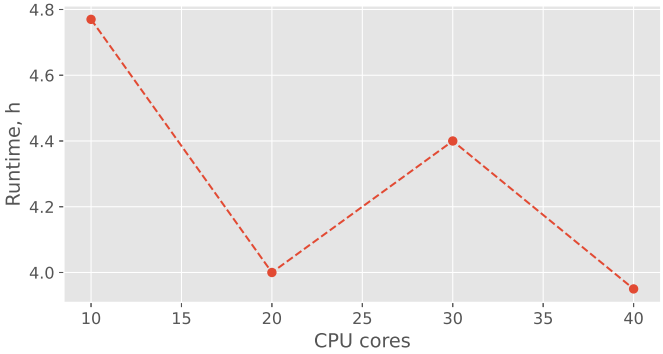
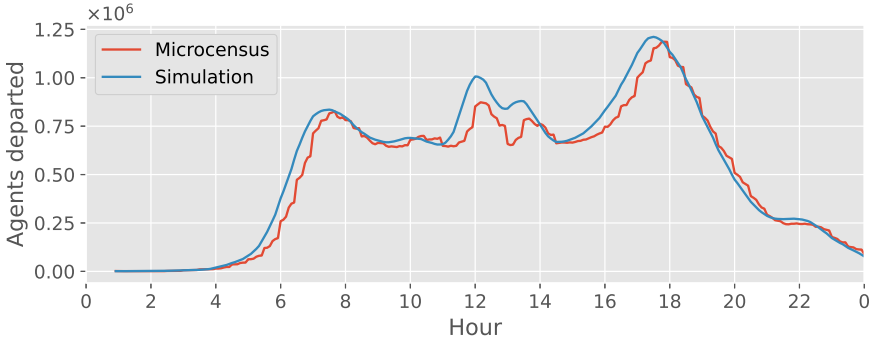


FIGURE 4.9: Runtime performance of the unified modelling pipeline for the Switzerland scenario on multi-core CPUs.

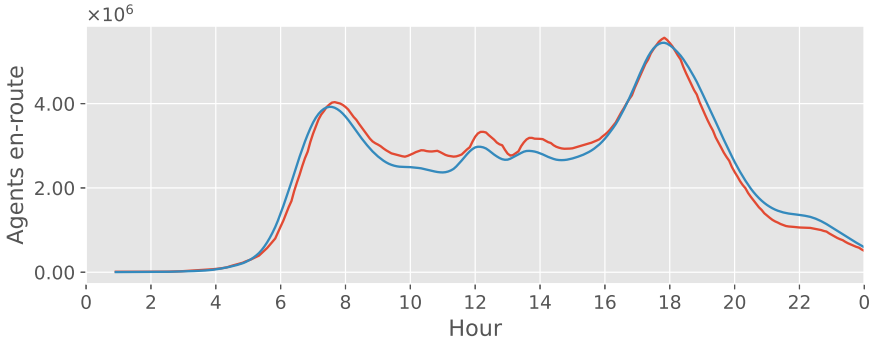
The unified modelling pipeline was used to generate the agent-based travel demand for a large-scale mobility scenario for the whole of Switzerland. The scenario was subsequently run with GEMSim. The scenario includes the entire population of Switzerland (3 million car drivers and 2.5 million public transit users and walkers) with their detailed travel demand; the road network (1.1 million links and 0.5 million intersections) generated from OSM; and the public transit schedule (30 000 stops and 20 000 routes) from SBB (Swiss Federal Railways), including routes for trains, buses, trams and other means of transport in Switzerland. The input synthetic population, as well as road network, are the same as was used for the older Switzerland scenario, shown in Figure 2.11. The scenario was run for 100 iterations to converge, and 10% of the agents were re-routed between the iterations.

Figure 4.10 compares the predicted departure and en-route dynamics to the micro-census. In both cases, the trends are captured, especially the dynamics of the morning and evening peak hours. The generated travel demand slightly overestimates departures in the afternoon, while the number of agents travelling simultaneously is underestimated. This can be explained by various reasons: some of the generated trips have shorter distances, traffic lights were not simulated in the scenario, or other factors such as pedestrians were not accounted for.

The scenario was validated against 291 traffic counts provided by the Federal Roads Office of Switzerland [177], which were previously used in Section 2.4, and the locations of the counting stations are shown in Figure 4.11. The counting stations cover the whole country and most of them are located along highways and primary roads. The peak morning (07:00–08:00) and evening (17:00–18:00) hours



(a) Departure dynamics.



(b) En-route dynamics.

FIGURE 4.10: Dynamics of the simulated agent-based scenario for Switzerland generated with the unified modelling pipeline, compared to microcensus data.

of a typical working day were used to compare simulated traffic flows with real data, as presented in Figure 4.12. The comparison shows agreement of simulation outputs with the field data, however, a few locations close to the country borders tend to be heavily underestimated as the cross-border traffic was not modelled.

Figure 4.13 compares the distributions of travel time and distance by simulated agents with the data from microcensus [278] for a typical working day of the week. For cars, the routed distance of trips is used (i.e., the total distance of the path to drive from one location to another, and not the Euclidean distance). For public transit, the Euclidean distance is used as the precise information about the routes of public transit vehicles is not always available. Moreover, public transit includes walking trips as a subset mode. Overall, the simulated results are in the agreement with the microcensus, and only some minor variations are present. For cars, short-distance trips are slightly underestimated, while long-distance trips are overestimated. One also needs to consider here that microcensus interviewees

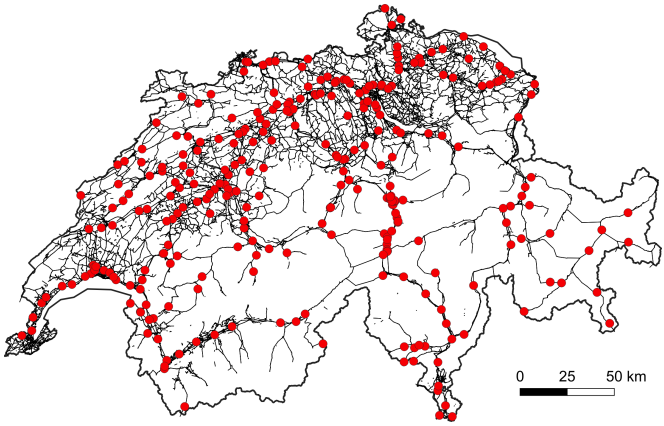
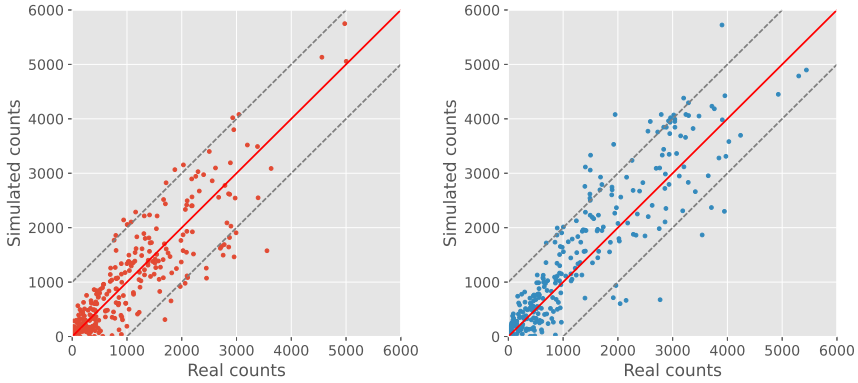


FIGURE 4.11: Locations of the traffic counting stations in Switzerland used for the validation of the unified modelling pipeline.



(a) Morning peak hour (07:00 – 08:00).

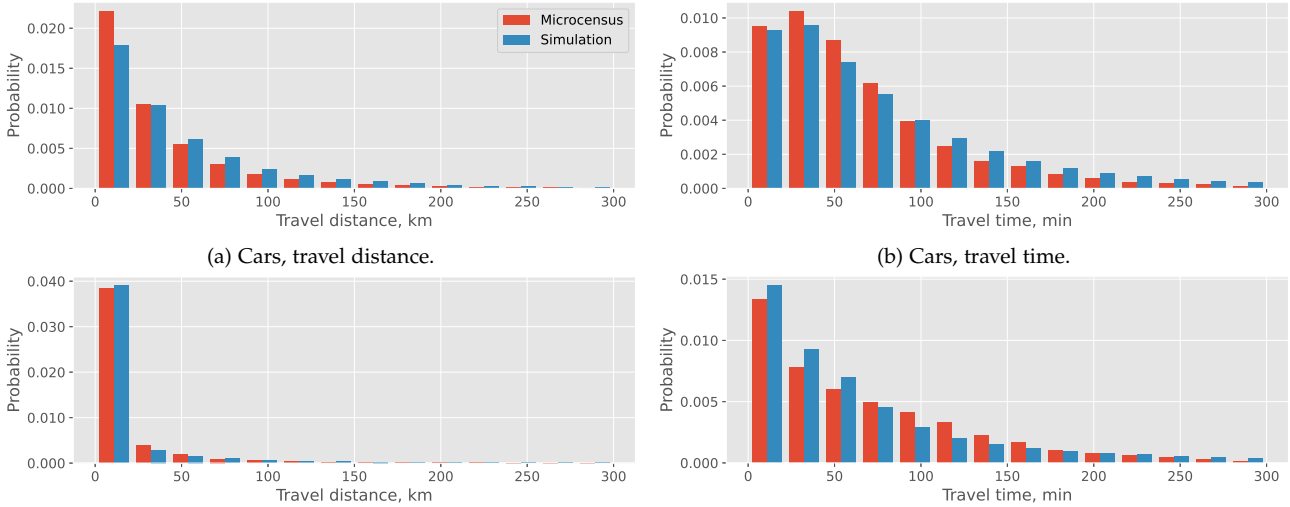
(b) Evening peak hour (17:00 – 18:00).

FIGURE 4.12: Comparison of simulated and real-world traffic counts in Switzerland using the unified modelling pipeline.

tend to round their answers regarding departure and arrival times, most often to 15 minutes. This rounding can impact the distribution of probabilities for travel times. For public transit, short-distance trips are slightly overestimated, while long-distance trips are underestimated. This can be affected by walking activities modelled together with public transit.

A similar comparison of distributions of travel time and distance, but for a typical weekend, is shown in Figure 4.14. Again, the results show an overall agreement of simulated data with the microcensus, and minor variations in distributions are similar to variations in respective distributions for a typical working day. This

comparison of the weekend scenario, however, must be taken with caution, as it may not be fully representative (for example, in terms of the daily dynamics of travelling) and is provided mostly to demonstrate the capabilities of the pipeline. The reason is that while during the working week people tend to demonstrate reasonably uniform day-to-day behaviour, this does not occur across weekends. Therefore, the presented pipeline can generate only averaged weekend behaviour due to the low availability of the weekend-related travel data. Nevertheless, should more weekend-related behavioural data become available, the pipeline can be utilized to generate custom scenarios for weekend days.



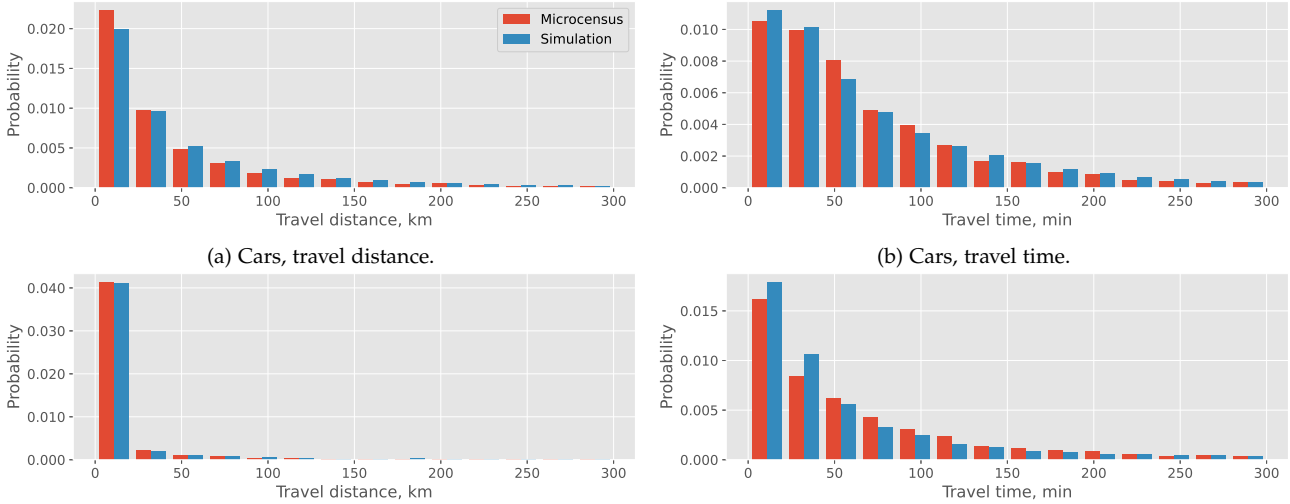
(a) Cars, travel distance.

(b) Cars, travel time.

(c) Public transit, travel distance.

(d) Public transit, travel time.

FIGURE 4.13: Distributions of travel time and distance for a typical working day in Switzerland generated with the unified modelling pipeline, microcensus and simulation.



(a) Cars, travel distance.

(b) Cars, travel time.

(c) Public transit, travel distance.

(d) Public transit, travel time.

FIGURE 4.14: Distributions of travel time and distance for a typical weekend in Switzerland generated with the unified modelling pipeline, microcensus and simulation.

5

MOBILITY-AS-A-SERVICE: MUNICH CASE STUDY

He who must travel happily must travel light.
— Antoine de Saint-Exupery

The design and implementation methodology for the integration of DRT services into public transit systems has been developed since the 1970s–1980s [287–289]. In later years, the methodology was improved with various concepts, like mixing fixed-route scheduled public transit services for high-density areas with DRT services on low-density areas [290, 291]. One of the problems with public transit is the first and last mile problem, when people do not use public transit due to long walking times at the beginning or end of a trip. This is the area where DRT services are considered to contribute positively. In reality, however, implementation of such hybrid systems faced many difficulties, including high operating costs and communication problems between drivers and riders. In recent years, with the development of global communication systems through the Internet, as well as the widespread adoption of mobile phones, novel business opportunities have appeared in the area of DRT services. Companies like Uber and Lyft offer flexible and relatively cheap DRT services including ride-hailing and ride-sharing.

Moreover, considering the anticipated rate of improvement of AVs, the operation of DRT services shall become even cheaper in the future as fleet operators can cut costs through reduced operating expenditures (no need to pay salaries and social insurance for drivers, better control of the fleet, optimization of parking and charging fees). In addition, as booking platforms are working online and the data about customer behaviour is constantly collected, operators can utilize ML algorithms to match the demand more efficiently using dynamic pricing schemes.

While ride-hailing and ride-sharing services can already provide competitive pricing schemes for customers when compared to other DRT services like taxi cabs, researchers and politicians have raised concerns regarding the impacts of companies like Uber on the local public transit systems. Typically, local public transit systems are subsidized with public funds to increase ridership and, in turn, to increase the frequency of service and reduce waiting times. In addition, shifting people to public transit can reduce negative externalities like traffic congestion and air pollution. Subsidies can also provide welfare redistribution effects by providing access to transport systems to low-income people [292, 293].

Some works indicate that DRT services lead to a reduction in public transit ridership [294, 295]. Interestingly, when a ride-hailing company starts operating in an urban area, it initially leads to the increased usage of public transit, thereby complementing it. But when another ride-hailing company enters the market in the same urban area, ride-hailing services start competing with public transit, and this

leads to the overall decrease of transit ridership below the level normal before the first ride-hailing service started operations. Other works show that DRT services can complement public transit [296, 297] instead of becoming a main mode of transport by providing a supply of riders. This, however, may require significant investment into infrastructure close to public transit lines. Ride-hailing, while decreasing usage of one transport mode like buses, can lead to the increased use of other public transit modes like rail [298].

There are further negative externalities to which DRT services can contribute, such as increased VKT [299], which typically leads to increased traffic congestion and air pollution, unless the average occupancy of the trips is increased through ride-sharing. By replacing walking activities that people might take when using public transit, DRT services may lead to negative health impacts like obesity [300]. The legal status of drivers working with ride-hailing platforms also remains unclear, and is a source of controversy in many countries [301]. Ride-hailing companies typically deny employee status, meaning that drivers do not get social insurance or other employment protection mechanisms. This means that social costs are not carried fully by companies, which can lead to biased distribution of social welfare.

Overall, controversy surrounds the business of ride-hailing companies. DRT services can offer positive experiences to some people but can also cannibalize public transit systems with successive deterioration of their quality. A proper balance that maximizes the overall social welfare is to be generated in the regulation of ride-hailing companies. Moreover, the impacts of DRT services may differ depending on their geographical context. For example, in the USA, public transit systems have historically low coverage and a small percentage of the overall mode share split, as people tend to use cars. In many European countries, public transit systems are more widely available, and cities target the reduction of cars through the extensive development of public transit. Therefore, the impacts of integrating ride-hailing with public transit systems should be estimated in advance and take into account local specifics.

Recently, the concept of MaaS has gained a lot of attention and development. MaaS aims to provide the seamless integration of multiple modes of transport, including public and private providers of services, giving users maximum flexibility in trip planning. Moreover, with MaaS, a user does not need to own access to certain transport modes, but they can use a pay-as-you-go principle, when a MaaS provider has a unified and convenient gateway for trip payment through a single digital application. Therefore, instead of having fixed subscriptions for transport modes, a user can obtain travel services based on their needs.

One way to estimate the integration of DRT services with public transit systems is agent-based modelling. GEMSim's ability to model public transit and coordinated fleets was demonstrated in Chapter 3, and a flexible architecture of the simulation framework allows the coupling of both modes to study even more complex cases. This chapter contributes to the area of MaaS, which includes public transit and a coordinated fleet used for DRT services in the Munich metropolitan region (Germany). In the example adopted for this study, a MaaS provider operates a taxi

fleet used as a demand feeder for the public transit system by serving the first and last mile travel legs of agents' trips. The demand-supply interaction makes it possible to estimate the users who may benefit from the proposed MaaS, and examine the causal effects of agents' decisions.

5.1 BACKGROUND

As with the impact of DRT services on public transit, studies on transport systems with integrated public transit and DRT services show varying results. Oh et al. [302] studied the integration of automated mobility on-demand (AMoD) services into Singapore's public transit system using agent-based scenarios with various adoption levels of AMoD. In the moderate adoption scenario, the share of AMoD in the modal split reaches 5.8%–8.9% depending on the pricing scheme, while the share of AMoD combined with public transit is only 1.6%–2.3%. At the same time, VKT increases by 11%–17% together with congestion, which increases by 14% in the morning peak hour. In the high adoption scenario, AMoD share reaches 11.9%–18.8%, while AMoD combined with public transit gets up to 3.4% of the model split. At the same time, VKT increases by 25%–42%; so too does morning peak hour congestion, by 28%. The authors conclude that AMoD leads to the cannibalization of public transit unless mitigated with policies (e.g., discounts for trips, pricing schemes, more limited area-wide deployment of fleets, etc.).

Basu et al. [303] evaluated two agent-based scenarios with AMoD in a virtual city of 351 000 inhabitants and Singaporean activity patterns: replacement of buses and mass rail transit (MRT), and the integration of AMoD with public transit. The replacement of public transit with AMoD leads to much higher congestion levels compared to the scenario in which AMoD complements it, especially during off-peak hours. In overall, AMoD gets 16.7% in modal shares for the replacement scenario and 6.6% when complementing public transit. In terms of the customer perspective, the replacement scenario leads to the most unfavourable travel experience with longer waiting and in-vehicle times due to higher congestion levels. In both scenarios, the total VKT is increased, and fleets generate about 40%–45% of empty driven distance. Moreover, VKT may increase even further with the increased travel demand for AMoD despite using ride-sharing.

Scheltes and de Almeida Correia [235] performed a case study on the connection between the campus of TU Delft university and the train station Delft Zuid (Netherlands) using AVs for the last mile problem. Trip lengths were distributed between 1.5 km and 2.4 km, and there is no public transit between the locations; walking and cycling are used by about 5 000 people daily. Agent-based simulations showed that the proposed system was only able to compete with walking, not with cycling. In order to compete with cycling, additional measures for AVs are required. The authors also found that fleet re-balancing and pre-booking allow significantly reduced waiting times and reduction in average travel times. While

only a single-seat vehicle was used in the study, shared vehicles with larger capacity can bring more operating and economic advantages.

Gurumurthy et al. [304] studied the impacts of introducing a fleet of shared automated vehicles (SAVs) in Austin, Texas (USA). Multiple scenarios for fleet integration were evaluated, including door-to-door service, first and last mile travel legs, and the combination of the two. Scenarios use a 5% population sample with about 45 000 agents and a fleet of 4 500 SAVs, as well as the public transit network for the area. All scenarios showed a system-wide increase in VKT, and this increase is of at least 6% when low fares for SAVs are applied. When SAVs serve the first and last mile travel legs, VKT only increases by 1.6%. Low fares for SAVs impact transit ridership negatively by cannibalizing public transit, but high fares lead to increased transit ridership during peak hours when SAVs complement public transit. Low fares mostly impact the use of public transit in low-density areas, while high fares impact high-density areas.

Stiglic et al. [305] investigated the potential benefits for residents through the integration of public transit and ride-sharing services. Extensive computational experiments have been performed for an artificial urban area of 20 x 10 miles with an urban center radius of 2.5 miles, and which has two commuter train lines and four urban rapid transit lines. The study showed that an integrated transport system can significantly enhance mobility and increase the use of public transit. For example, the average length of detour time for ride-sharing drivers is 7.2% of the total distance when integration with public transit is implemented, compared to 8.4% when no integration is provided. The authors also emphasize that public transit frequency is more important than speed to provide better performance for ride-sharing fleets.

Shen et al. [236] proposed and evaluated an integrated system with public transit and DRT services in the Tampines area of Singapore with about 240 000 inhabitants living in an area of 12 km^2. A fleet of AVs was used to serve the first and last mile travel legs to local MRT lines, and the fleet was fully integrated into the public transit system in terms of fares, tickets and information. High-demand buses were kept in the study, while low-demand routes were replaced with introduced DRT services. The results show that the proposed system is financially sustainable, improves the quality of service and leads to lower usage of road resources, especially when AVs are shared. The study, however, assumes static AV demand and does not account for demand-supply interaction.

Huang et al. [306] presented a dynamic ride-pooling algorithm to match the demand of SAVs with the known schedule of trains in the area of central Austin, Texas (USA). The area was split into two automated mobility districts, served by different fleet operators during the 3-hour morning peak with a demand for public transit from about 4 000 riders, and the rest of the modes were not simulated. SAVs were used to serve the transit riders' first and last mile travel legs. The study showed the importance of public transit schedule in ride-pooling algorithms in order to provide higher quality service. Only about 57% of transit riders could catch a train on time when no coordination with the train schedule was performed, compared

to about 87% when coordination was explicitly performed. However, coordination leads to about a 24% higher VKT compared to the baseline scenario with similar fleet size, as well as 48% longer waiting times and 86% longer trip durations.

Lau and Susilawati [307] used a four-step model of Kuala Lumpur (Malaysia) to study the integration of SAVs for transit riders' first and last mile travel legs in the morning peak hour. The area around MRT and LRT (light rail transit) of Kuala Lumpur was considered, as it contributed about 40% of total passenger demand in the city. The routes for SAVs were pre-assigned with pickup and drop-off locations around stops of public transit. Hence, the focus of the study was on passengers who drive cars to the nearest LRT and MRT stations. The results show that public transit usage increased by 3% while the total VKT of personal cars dropped by 6%. However, the authors emphasize that the reduction in waiting time for SAVs by about 20% leads to a significant decrease in public transit as more people tend to switch to SAVs.

Wen et al. [238] studied the transit-oriented deployment of SAVs with integrated demand-supply interaction. The study area of 15 x 10 km with 159 000 residents was located in a major European city with an extensive and developed public transit system (45% of mode share). A fleet of SAVs was used to serve the agents' first and last mile travel legs. No traffic was simulated, and only static travel times were used. The results show that a fleet size represents a balance between the quality of service and operating costs. Trip sharing increases the efficiency of operation and reduces the costs; however, it also leads to a 10%–15% increase in in-vehicle time for passengers. Fare discounts impact the choice of SAVs and require better integration into the system. Interestingly, for trips downtown, 73% of SAV passengers are from rail, 17% from park-n-ride and 10% from a car mode. For intrazonal trips, 82% of passengers come from a car and 18% from a bus mode.

Overall, many previous studies used simplified assumptions for some parts of their scenarios, such as not accounting for traffic congestion, using static demand for DRT services, not including demand-supply interaction, using artificial areas for case studies, or limiting the possibilities of agents to decide when to use DRT services. Moreover, none of the papers assessed the impact of geo-fencing for the first and last mile travel legs, while the impacts of public transit accessibility on the use of DRT services in the area were not well studied. Moreover, the spatial scale for most of the performed simulations was relatively small, so it would be interesting to see how the proposed integration of DRT services with public transit is affected by the deployment scale.

5.2 SCENARIOS

This case study uses the Munich scenario, which covers the whole of the Munich metropolitan region with the agglomerated areas of Munich, Augsburg, Ingolstadt, Landshut, Rosenheim, and Landsberg am Lech. The region's total population is about 6 million inhabitants, the distribution is shown in Figure 5.1, and the

region covers about 40% of the area of Bavaria. The region's road network, which includes 289 893 nodes and 840 752 links, was generated from OSM. The public transit schedule was taken from the OpenData ÖPNV portal [308], and includes all provided 1 555 lines and 34 206 stops in the region. The initial daily-activity plans of the agents were generated based on the methodology described in Chapter 4 and using the Mobility in Germany [309] nationwide travel survey. The synthetic population includes 3 248 739 agents who either drive cars or use public transit; walking was considered a sub-mode of public transit. People who either stay at home or use other modes, such as bicycles, were not included in the simulation.

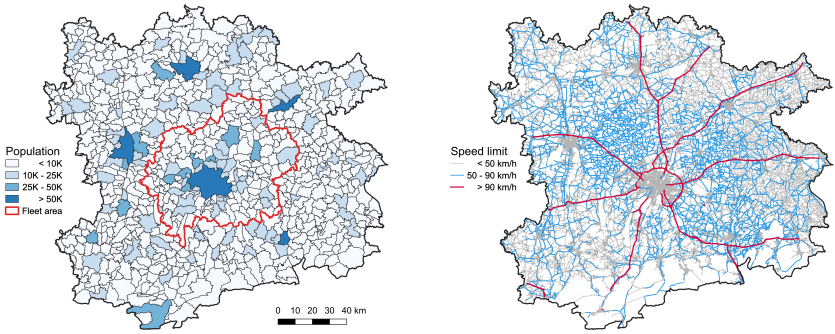


FIGURE 5.1: Distribution of potential travel demand (left) and road network (right) in the Munich metropolitan region.

The original generated scenario was run for 400 iterations to converge the traffic. Moreover, compared to the scenario of Switzerland, 10% of agents, in addition to 10% of re-routed agents, were allowed to switch a transport mode. There are two reasons for the mode change strategy to be permitted in the Munich scenario. First, the data quality for Germany is lower than for Switzerland; for example, trip data is given at the municipality resolution level, what makes it difficult to evaluate the effect of public transit quality on mode choice. Second, as one of the goals of the present case study is to evaluate the impacts of coordinated fleets integrated into public transit systems as MaaS, in the reference scenario agents should be allowed to pick an optimal transport mode in order to later distinguish the effects of fleets rather than a simple mode choice estimation.

The converged baseline scenario was used to run scenarios with a MaaS mode that includes DRT services integrated with public transit. The DRT operation area, where taxi vehicles are allowed to pick up and drop off passengers, is shown in red on the left side of Figure 5.1. The fleet operation area includes the city of Munich and surrounding neighbourhoods of varying population density. While the fleet operates in a limited area, the whole Munich metropolitan region was simulated with car traffic and public transit in order to keep boundary conditions more realistic.

Fleets of 5 000, 10 000, 15 000 and 30 000 vehicles were simulated based on the reviewed literature that suggests some optimal fleet sizing depending on potential demand. As the demand for DRT is not known in advance, four sizes were simulated for each set of fleet operating policies. In general, for the fleet operation, the same assumptions used in Chapter 3 were kept. In addition, simulated DRT service has some limitations imposed to prevent agents using it as a personal taxi without the use of public transit:

- Pickup and drop-off locations can be chosen either at origin and destination points or at public transit facilities like train and bus stops.
- DRT services are spatially limited and a taxi can drive a passenger only within an area of a certain radius, called a DRT lookup area, around the origin and destination points of a trip, while the rest of the trips are covered by public transit or walking.
- In general, trip's DRT lookup areas should not intersect; therefore, the minimum distance of a trip is as twice as the limiting radius of DRT service.
- When the intersection of DRT lookup areas is explicitly allowed, locations within the intersected area are not considered by DRT service. This should not allow agents to make a whole trip consisting of two DRT travel legs with the transfer location in the middle.
- First and last mile travel legs shorter than 700 m are covered by walking and DRT service does not operate over these short distances.
- Public transit is only used when the expected duration of a trip without using DRT service is shorter.

In GEMSim, the integration of public transit with DRT services was implemented within a separate MaaS module that provides intermodal routing considering public transit schedule and expected waiting times of DRT services. Multiple DRT operators are supported, with or without overlapped operation areas; as a result, an agent may use one operator for the first mile travel leg and another for the last mile travel leg. The functionality of the proposed MaaS solution is available for agents through a dedicated transport mode, which works in a way similar to cars or public transit, simply by specifying the mode for a travel leg in an agent's daily-activity plan. In the present case study, in addition to different fleet sizes, multiple DRT radii for DRT lookup areas were evaluated: specifically, 1.5 km, 3 km and 5 km. Intersection of DRT lookup areas was allowed for the scenarios with 5 km and 3 km radii, while the 3 km radius was also used in scenarios with and without intersecting lookup areas. In total, 16 scenarios were run, for each combination of the fleet size and the radius policy for DRT lookup areas. Each scenario was run for 400 iterations to converge: in the first 350 iterations, 10% of the agents were re-routed and the other 10% were allowed to change the trip mode between a car, public transit and the

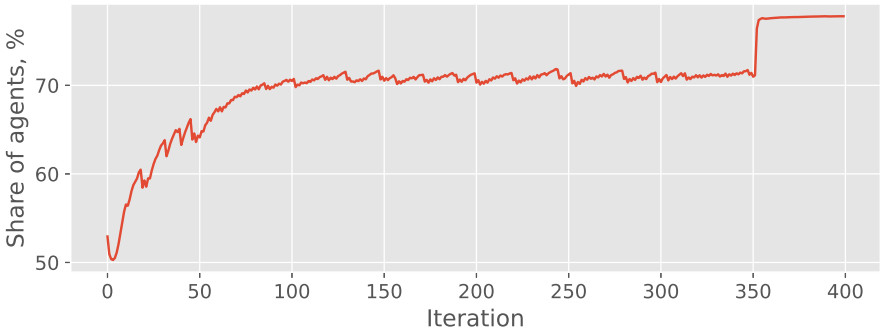


FIGURE 5.2: Share of agents using a car in the Munich region while converging the baseline scenario.

newly introduced MaaS mode. Agents who make trips during the day outside of the fleet operation area were not allowed to change the mode to MaaS, only between a car and public transit. In the last 50 iterations, agents stop innovating their plans and choose only from their previous memory of five plans.

5.3 RESULTS

5.3.1 *Convergence*

Figure 5.2 shows how the share of agents using a car changes between the iterations in the baseline scenario. Initially, about 53% of agents have a car assigned as a main transport mode, and 47% of agents have public transit. However, according to the German mobility survey data, only about 10% of the trips are performed by public transit and about 22% of the trips are made by foot. As public transit and walking trips are combined in a single mode, and many people who use public transit also perform walking trips, in the Munich scenario, the actual number of agents using a car can be larger. Hence, the share of agents using public transit and walking should be around 25%–30%. These numbers fit well with the simulation results, where more agents switch from public transit to a car.

Figure 5.3 shows the average score of the agents in the baseline scenario. It rapidly increases in a few iterations as the traffic congestion dissipates and the agents learn which modes provide a better travel experience. Most of the score advancement happens in the first 50 iterations, when agents actively change their modes. After the mode change rate drops, the improvement in score also stops increasing. One can note, however, that the score has sharp drops in some iterations, which corresponds to situations when many agents did not choose an optimal mode. With a non-optimal mode, an agent can travel long times either due to the poor quality of the public transit system in the area, or due to increased traffic congestion when many

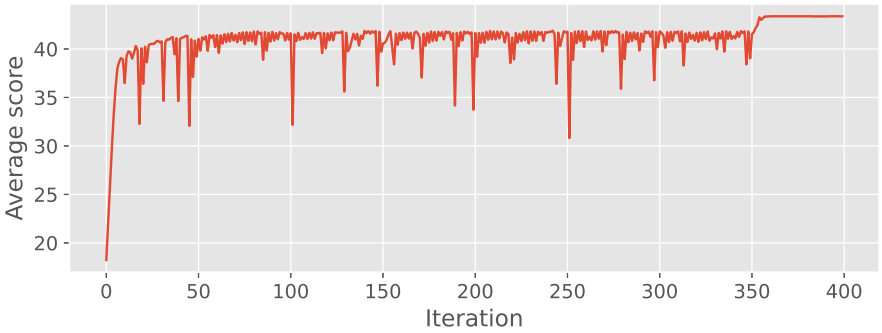


FIGURE 5.3: Average score of agents in the Munich region while converging the baseline scenario.

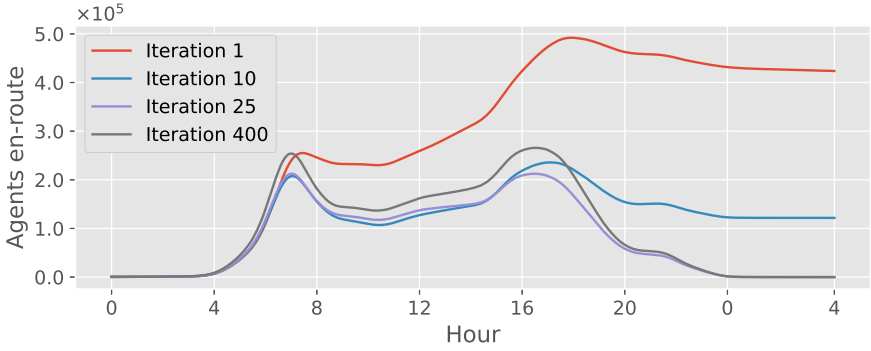
agents in the same area switch to a car. After 350 iterations, there are no score drops as agents choose from modes and routes learned from their previous experience.

The dynamics of en-route agents by car and public transit during the day for the baseline scenario are shown in Figure 5.4. There are two main travelling peak hours, starting from 07:00 in the morning and from 17:00 in the evening, for both the car and public transit modes. One can note how the convergence evolves throughout the iterations. In a few iterations, many agents are either stuck in congested and gridlocked traffic, or cannot find suitable routes using public transit. In later iterations, more agents switch from public transit to the car mode, so congestion on the roads after 25 iterations increases at the end. It is also important to note that almost 25 000 agents, who had problems with public transit by the end of the day, switched to the car mode. Overall, the agents learned which transport modes satisfy their daily-activity needs and how to drive through the traffic, and then switched towards favourable solutions.

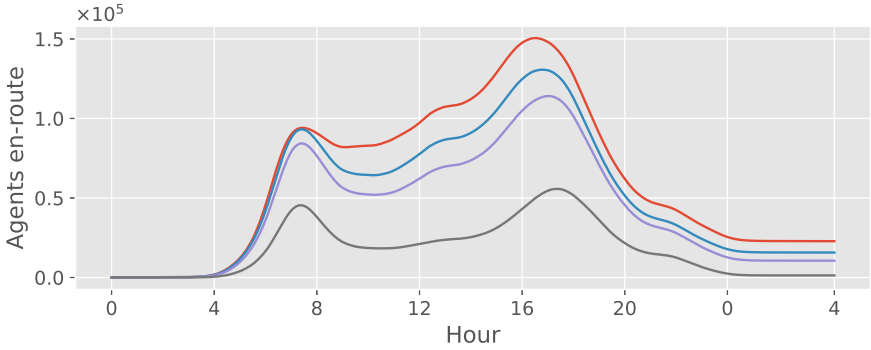
5.3.2 *Validation*

The baseline scenario was validated with traffic counts provided by the Federal Highway Research Institute of Germany [310]. Data for a random working day from the year 2017, which was when the German travel survey was conducted, was used for the validation. Figure 5.5 shows the locations of 64 traffic counting stations in the Munich metropolitan region; most of the stations are located along major highways and around the city of Munich.

The comparison of simulated traffic counting data with real measurements for the periods from 07:00 to 08:00 and from 17:00 to 18:00 is shown in Figure 5.6. Simulation results are in agreement with real data and the trends are captured. Simulation tends to underestimate traffic counts at some locations, mostly at the



(a) Agents travelling by car.



(b) Agents travelling by public transit.

FIGURE 5.4: Agents en-route during the day in the baseline Munich scenario.

border of the case study area. This underestimation is expected, as the behaviour of people from neighbouring municipalities was out of the scope of this study.

5.3.3 *Fleet performance*

Table 5.1 shows the overview of simulated scenarios in the Munich area. The scenario names are abbreviated in the following way: first, a DRT lookup radius in kilometres is specified; second, a fleet size in thousands follows the underscore symbol; lastly, the *i* suffix is specified in case intersection of DRT lookup areas was allowed. Here, N_{req} is the number of served DRT requests by a fleet operator, and D_{req} is the average Euclidean distance, in km, between the pickup and drop-off locations of the requests.

The first interesting observation is how the mode shares change in the scenarios. In general, the share of car users is close to its baseline value of 78%, dropping

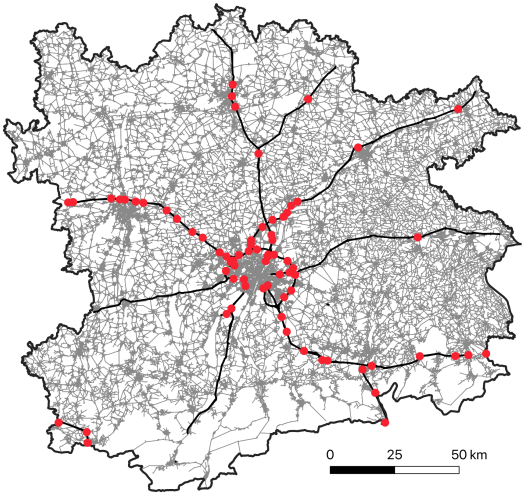
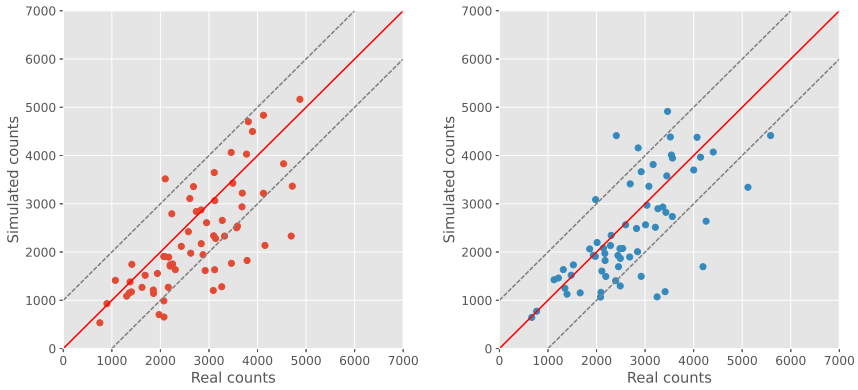


FIGURE 5.5: Locations of the traffic counting stations in the Munich area used to validate the baseline scenario.



(a) Morning peak hour (07:00 – 08:00).

(b) Evening peak hour (17:00 – 18:00).

FIGURE 5.6: Comparison of simulated and real-world traffic counts in the Munich area.

to 72%–75% only when fleets of 30 000 vehicles are used. On the other hand, the share of public transit drops significantly in all scenarios with MaaS, meaning that most of the MaaS users come from public transit. Similar behaviour has been observed in other studies in which automated fleets cannibalized public transit services. However, one should treat these numbers with care as the simulation generated a probability that agents can pick a non-optimal mode. Figure 5.7 shows the distribution of daily travel time changes for agents who switched from cars to

TABLE 5.1. Overview of simulated MaaS scenarios in the Munich area. VKT and VHT are given for fleets.

Scenario	Mode share, %			VKT	VHT	N_{req}	D_{req}
	Car	PT	MaaS				
Baseline	78.00	22.00	–	–	–	–	–
r1.5_5k	75.86	13.35	10.79	1 549 420	65 059	361 129	1.20
r1.5_10k	75.37	13.61	11.02	2 510 495	108 867	605 023	1.20
r1.5_15k	75.28	13.86	10.86	2 669 019	116 365	645 551	1.20
r1.5_30k	74.24	13.88	11.88	3 106 329	117 697	896 297	1.20
r3.0_5k	77.48	14.91	7.61	1 658 872	67 600	258 123	2.13
r3.0_10k	76.95	15.51	7.54	2 863 073	120 474	438 161	2.20
r3.0_15k	76.61	15.80	7.59	3 241 530	137 146	499 658	2.23
r3.0_30k	75.77	15.91	8.32	3 883 052	152 538	660 902	2.23
r3.0_5ki	74.40	10.62	14.98	1 911 159	75 727	274 826	2.18
r3.0_10ki	74.44	10.98	14.58	2 998 306	123 594	477 046	2.19
r3.0_15ki	74.18	11.27	14.55	3 927 377	161 613	648 355	2.24
r3.0_30ki	72.92	11.93	15.15	5 308 567	208 122	902 361	2.27
r5.0_5ki	75.02	10.79	14.19	1 845 870	71 887	209 492	3.51
r5.0_10ki	74.78	10.93	14.29	3 244 051	130 225	371 979	3.46
r5.0_15ki	74.42	11.27	14.31	4 361 780	170 054	502 337	3.52
r5.0_30ki	72.72	12.23	15.05	6 737 799	261 141	753 804	3.63

MaaS. This was from one of the iterations from the scenario with a DRT lookup radius of 1.5 km and a fleet of 15 000 vehicles; in other scenarios, the output is similar. Positive numbers mean that travel time increased, and for most of the agents this increase is quite substantial, with an average of 55 minutes (vertical dashed line).

This result shows that in reality, it is highly unlikely that people would abandon cars in favour of public transit or even MaaS as there is no incentive in terms of travel time. Hence, actual MaaS mode shares are expected to be 2%–3% lower due to the lack of time benefits for car users. In contrast, transit users can get travel time benefits when switching to MaaS, as shown in Figure 5.8. Here, most of the switchers get reductions in travel time, and even when someone experiences an increase in travel time it is mostly in the range of 0 to 20 minutes, which can be considered acceptable if split across multiple trips during the day.

Another interesting finding from Table 5.1 is how the structure of demand changes with the increase of the fleet size while the DRT lookup radius (and the allowance of intersection) is kept constant. For each combination of radius and the intersection allowance policy, while the share of MaaS users remains within the same narrow

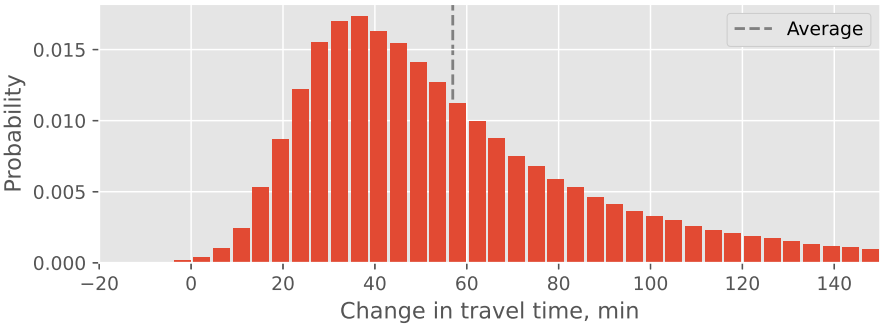


FIGURE 5.7: Change in daily travel time of agents in the Munich area who switched from a car to MaaS, scenario with 1.5 km DRT lookup radius and a 15 000-strong fleet.

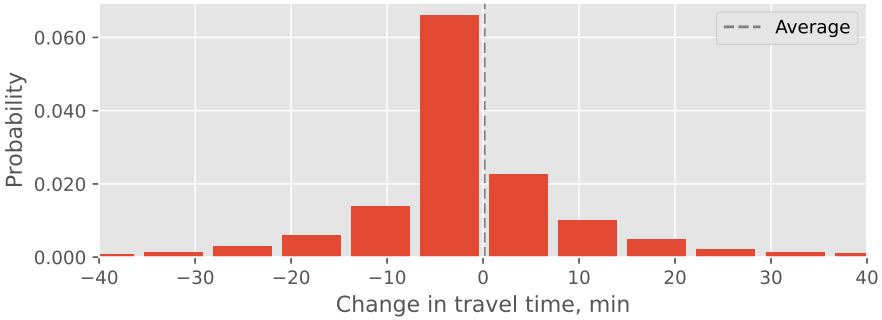


FIGURE 5.8: Change in daily travel time of agents in the Munich area who switched from public transit to MaaS, scenario with 1.5 km DRT lookup radius and a 15 000-strong fleet.

range, the number of served requests increases along with the fleet size. At the same time, the average distance per served request remains almost the same. This means that with the increase of the fleet size the demand shifts towards agents who can benefit from taking multiple DRT travel legs per day. For small fleets of 5 000 vehicles, one or even fewer DRT requests per agent who has switched to MaaS is served; for larger fleet sizes the number of requests per agent reaches values of around two. It should also be noted that even when an agent switches to MaaS mode, there is a probability that the whole trip will be performed with public transit if the waiting time for DRT services outweighs other options.

One can also note that with the increase of DRT lookup radius, VKT and VHT significantly increase for the same fleet sizes. This can be explained by the fact that for agents who travel longer distances with public transit, potentially making more transfers and walking, DRT can provide a larger improvement in their daily score, thus making the probability of switching to MaaS higher compared to agents with

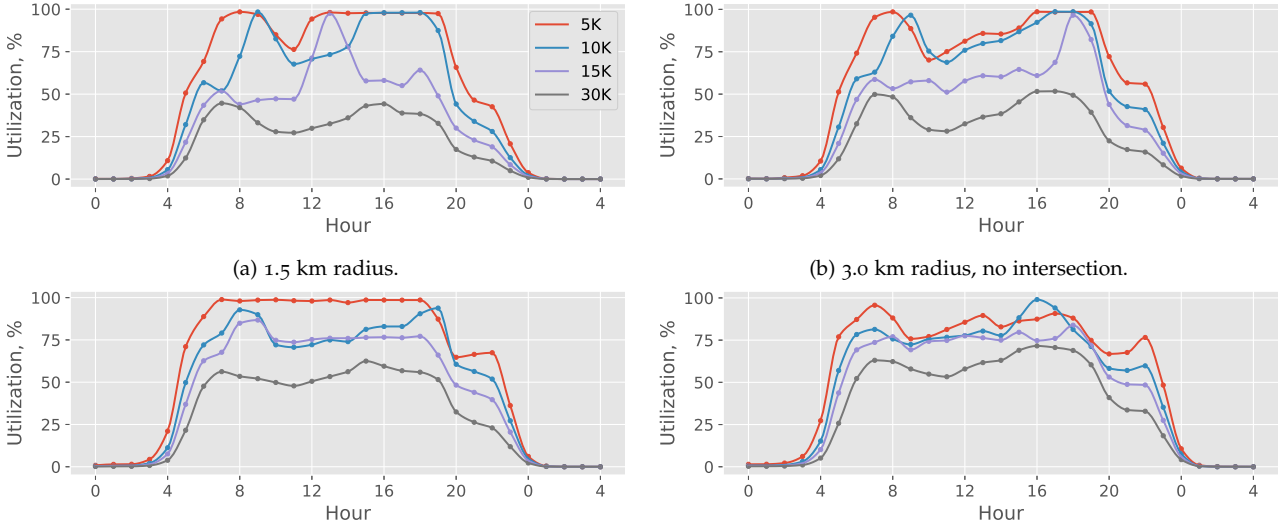
less benefits in travel time. This is also aligned with the increase of average distance per DRT request.

Fleet utilization, i.e., when the vehicles are not idle, is presented in Figure 5.9. The first trend is that increasing the fleet size leads to decreased utilization; this was expected. Large fleets of 15 000 and 30 000 vehicles are mostly underutilized throughout a day, while smaller fleets provide better utilization. The second trend observed is that, in general, increasing the radius of the DRT lookup area leads to higher fleet utilization for both small and large configurations. Notably, for the set of scenarios with 3 km radius allowing intersection of DRT lookup areas, the fleet of 5 000 vehicles is fully utilized from 07:00 to 18:00. The increase of utilization is expected with the increase of the radius as longer distance rides are performed.

A similar analysis was performed for the average waiting time of the passengers, shown in Figure 5.10. Here, as expected, larger fleet configurations provide shorter waiting times and hence a better quality of service. Fleet sizes of 15 000 vehicles and fewer are more sensitive to demand peak hours in the morning and in the evening, while a fleet of 30 000 vehicles provides almost uniform waiting times throughout a day. One can observe that a larger radius of DRT lookup area leads to increased waiting time for all fleet configurations. In general, when the radius is 1.5 km, the waiting time is in the range of 2–8 minutes for most of the day, and during demand peaks it increases up to about 20 minutes. Increasing the radius to 3 km raises the lower threshold of the waiting time to about 5 minutes, while peaks remain around the same values and only a fleet of 10 000 vehicles has a short peak of about 27 minutes in the morning. This morning peak in waiting time can also be the result of non-optimal and probabilistic behaviour of the agents during the simulations.

Interestingly, the allowance of the intersection of DRT lookup areas for a 3 km radius does not really impact the average waiting times, but helps to flatten both evening and morning peaks, as agents can find more optimal routes. At the same time, more agents may want to use the DRT service, hence, there is a higher chance that an idle vehicle will be closer to next requests. This higher usage of DRT services, when the intersection is allowed, is also observable in Figure 5.9. Further increase in the radius to 5 km leads to minimum average waiting time of about 10 minutes throughout a day, but with peaks of only around 16 minutes.

Empty driven mileage gives another insight into fleet performance, and is shown in Figure 5.11. First, smaller fleet sizes and smaller DRT lookup radii lead to higher empty mileage driven, causing them to contribute more to negative externalities like traffic congestion, air pollution and noise. For example, for radii less than 5 km a fleet of 5 000 vehicles constantly yields empty mileage of about 40%, with demand peaks around 60%. However, for a short DRT lookup radius of 1.5 km, even a fleet of 10 000 vehicles yields on average about 30% of empty mileage. Second, a small fleet of 5 000 vehicles has almost constant empty mileage above 40% in cases of 3 km radius with intersection allowed. This explains high daily utilization of this fleet configuration, as shown in Figure 5.9, and can be explained by increased demand as ride-hailing conditions become more flexible.



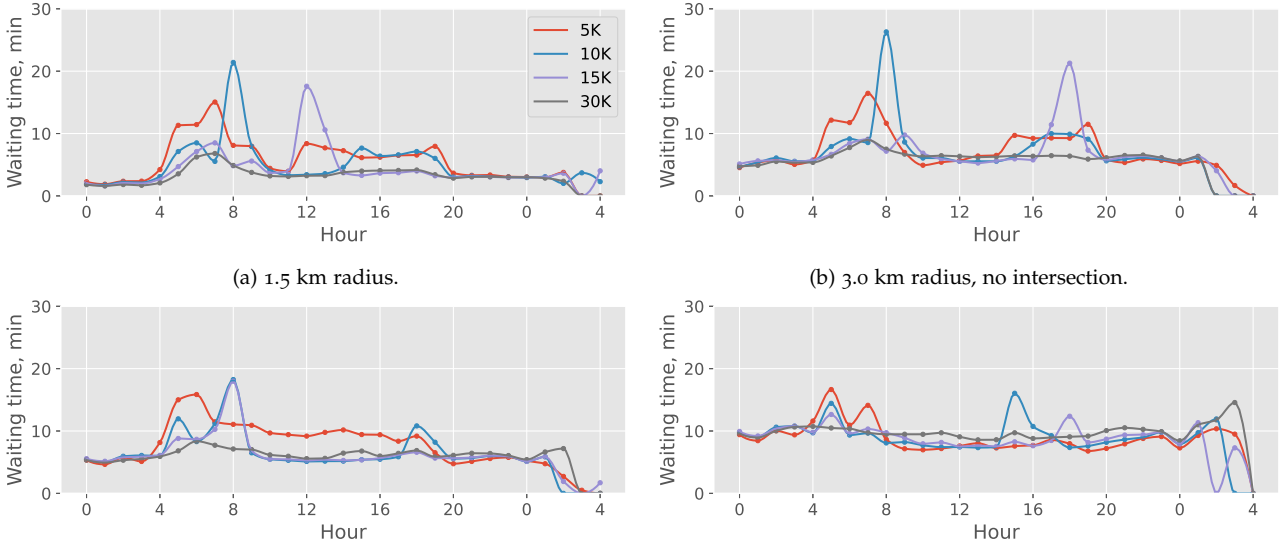
(a) 1.5 km radius.

(b) 3.0 km radius, no intersection.

(c) 3.0 km radius, with intersection.

(d) 5.0 km radius, with intersection.

FIGURE 5.9: Fleet utilization depending on the size and policy for DRT lookup in the Munich area.



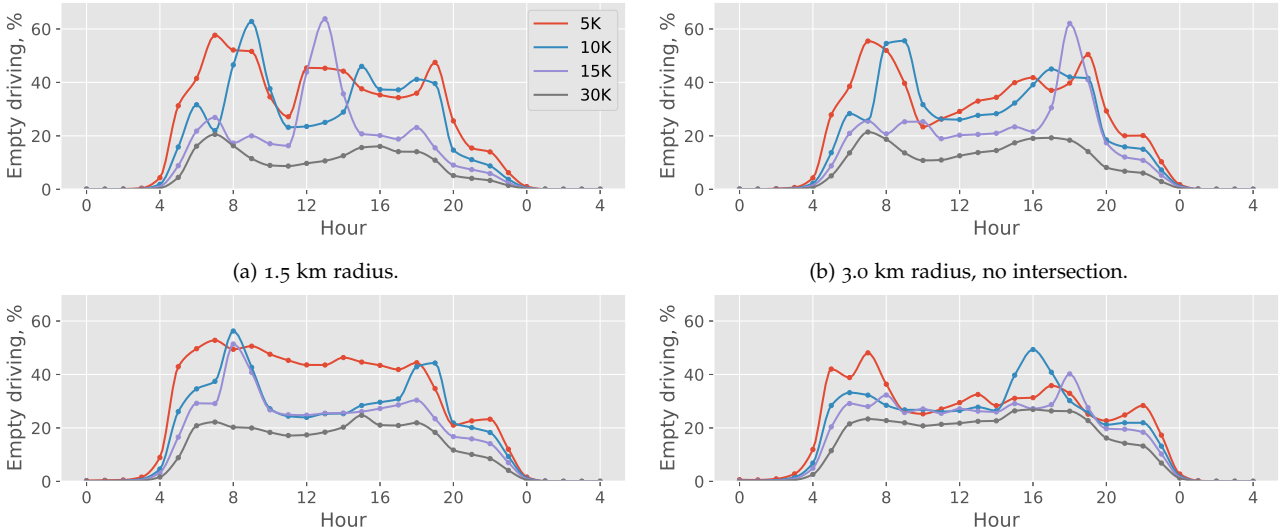
(a) 1.5 km radius.

(b) 3.0 km radius, no intersection.

(c) 3.0 km radius, with intersection.

(d) 5.0 km radius, with intersection.

FIGURE 5.10: Average waiting time depending on the size and policy for DRT lookup in the Munich area.



(a) 1.5 km radius.

(b) 3.0 km radius, no intersection.

(c) 3.0 km radius, with intersection.

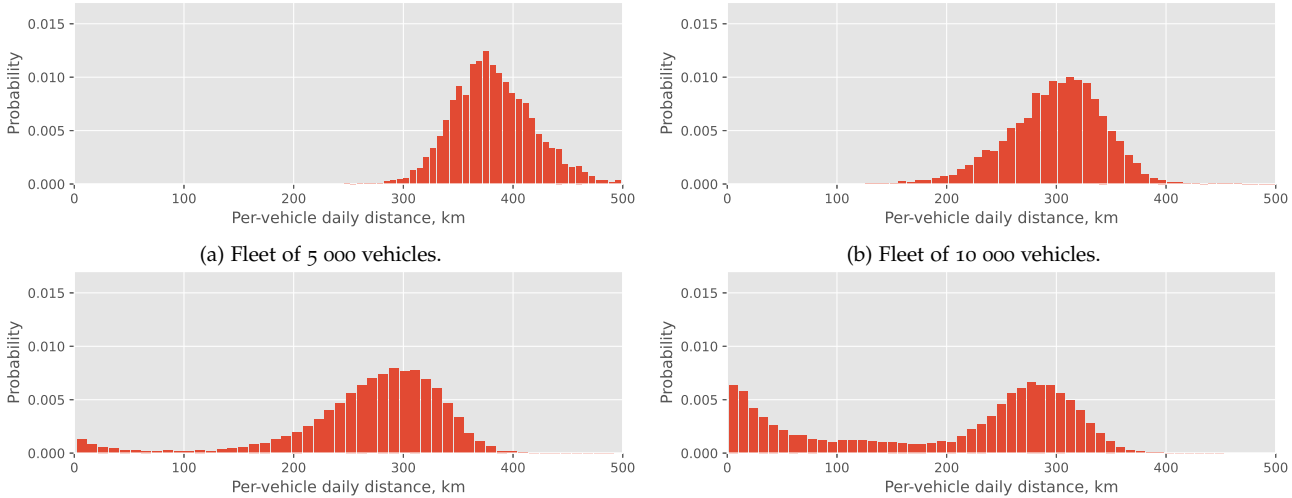
(d) 5.0 km radius, with intersection.

FIGURE 5.11: Empty driven mileage depending on the size and policy for DRT lookup in the Munich area.

Overall, the analysis shows that more flexible conditions for ride-hailing, in terms of larger DRT lookup areas and the possibility of their intersection at both ends of public transit trips, provide better fleet performance for operators and customers. From the operator's point of view, constant fleet utilization of around 75%–80% provides better demand predictions and leaves opportunity for additional optimizations, for example re-balancing or battery recharging during a day. Moreover, in the event of road accidents or fleet vehicle maintenance needs, the operator can redistribute the workload among the whole fleet. From the customer's point of view, predictable and short waiting times provide higher quality of service.

At the same time, when designing such integrated services, special attention should be put on negative externalities produced by fleets. For example, in the case of Munich, the increase of DRT lookup radius leads to the increase of VKT and VHT in almost every scenario due to the shift in the demand structure as the same number of agents tend to use DRT services more during a day. Therefore, limiting the distance of DRT trips can be one of the policies to regulate demand and keep negative externalities within an acceptable range. One can also note that increasing DRT lookup radius from 3 km to 5 km does not bring significant benefits, except when using the smallest fleet configuration, which becomes overutilized with 3 km radius and allowed intersection. Otherwise, the demand shifts to the travellers with longer distances, when a fleet is utilized more as a real taxi service instead of feeding public transit as a backbone mode.

The distribution of per-vehicle daily distances varies with the fleet size, and larger fleet configurations have lower average distances. The example of distance distribution for scenarios with the DRT lookup radius of 3 km and intersection enabled is shown in Figure 5.12. As expected, the increase in fleet size moves the distribution towards lower values, and for large fleet configurations the distribution transforms from a normal-like shape to a bimodal shape. One reason is that the larger the fleet, the higher the chance that an idle vehicle would be close to an incoming ride request, so some vehicles will have short pickup trips. These vehicles correspond mostly to the daily distance range of between 100 km and 200 km. Another reason is that large fleets have more underutilized vehicles serving low-demand areas, and these vehicles have daily distances below 100 km.



(a) Fleet of 5 000 vehicles.

(b) Fleet of 10 000 vehicles.

(c) Fleet of 15 000 vehicles.

(d) Fleet of 30 000 vehicles.

FIGURE 5.12: Distribution of per-vehicle daily distances for the scenario with a DRT lookup radius of 3 km with intersection in the Munich area.

5.3.4 *Public transit accessibility*

Another intention of this case study was to look into correlations between public transit accessibility and the use of MaaS. A public transit quality map for Germany has been created using the same methodology as for Switzerland [282]. As was described in Chapter 4, the area is divided spatially into four classes of public transit quality, from A (highest) to D (lowest), although areas may have no public transit quality classes as well. Table 5.2 shows public transit quality at the home locations of the agents who switched to MaaS. Most of the agents live in the highest classes, A and B; shares of the classes do not deviate much between the scenarios: 50%–55% of the agents are in class A, and 30%–32% are in class B.

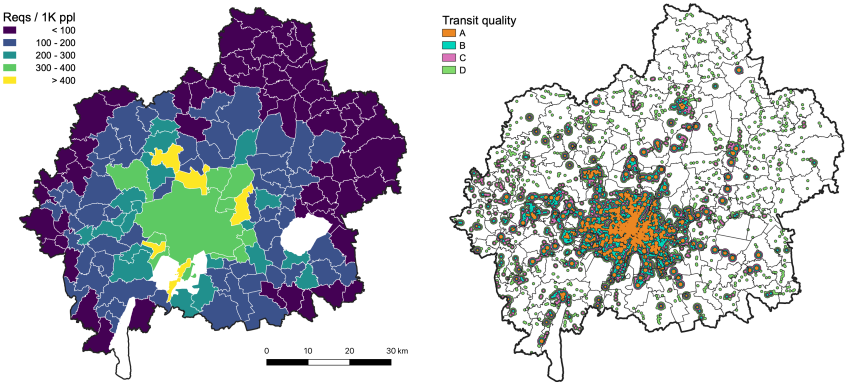
TABLE 5.2. Public transit quality at the home locations for agents who switch to MaaS in the Munich area.

Scenario	Class A, %	Class B, %	Class C, %	Class D, %	None, %
r1.5_5k	55.07	31.01	9.58	2.83	1.51
r1.5_10k	54.89	30.84	9.70	2.92	1.65
r1.5_15k	54.27	30.87	9.92	3.10	1.84
r1.5_30k	52.71	31.03	10.52	3.50	2.24
r3.0_5k	50.83	32.60	11.02	3.33	2.22
r3.0_10k	50.83	31.80	11.04	3.58	2.75
r3.0_15k	49.92	31.78	11.36	3.86	3.08
r3.0_30k	48.40	31.53	11.87	4.47	3.73
r3.0_5ki	53.28	30.89	10.39	3.25	2.19
r3.0_10ki	53.93	30.62	10.18	3.09	2.18
r3.0_15ki	53.83	30.35	10.21	3.22	2.39
r3.0_30ki	52.55	30.23	10.64	3.69	2.89
r5.0_5ki	51.23	31.57	10.99	3.61	2.60
r5.0_10ki	51.78	31.40	10.78	3.46	2.58
r5.0_15ki	51.94	31.16	10.73	3.49	2.68
r5.0_30ki	50.92	30.83	11.06	3.94	3.25

The distribution of public transit quality classes shows that about 80%–85% of potential MaaS adopters live in areas with high-quality public transit services, in terms of available transport modes and their frequencies. As demonstrated, the increase in DRT lookup radius does not lead to a higher share of MaaS, and the quality of public transit is one of the key factors. This is somewhat counterintuitive as DRT services are supposed to solve the problem of the poor connectivity of agents to public transit systems. However, when public transit does not operate with a certain level of quality in an area, DRT services become less useful because they cannot organise delivery of agents to major transport hubs. Considering the

additional waiting time for passengers, there would be little incentive for agents to use DRT.

Figure 5.13 shows (a) the density of DRT requests per thousand inhabitants in a municipality and (b) a public transit quality map of the same area. It is clear that more requests are served in the municipalities with a higher quality of public transit, which is primarily the city of Munich (with some other neighbourhoods). As soon as the quality of public transit drops when moving outwards from Munich, the number of DRT requests also drops significantly.



(a) Requests per 1 000 of population.

(b) Public transit quality.

FIGURE 5.13: Density of DRT requests (left) and public transit quality (right) in the Munich area.

Therefore, DRT services should not be considered as a standalone solution for the first and last mile problem. They should be developed together with public transit infrastructure in order to become attractive for at least existing public transit riders as part of potential MaaS packages. Considering Germany, Mocanu et al. [311] evaluated and compared accessibility by car and public transit for the whole country; the authors indicated that public transit is not a competitive alternative to cars in terms of travel times. Results showed that travel times for public transit are three times higher on average, and, moreover, this large discrepancy does not depend on region type. The authors concluded that, in order to motivate people to switch from cars to public transit, both pull and push measures are required: heavy investments into public transit infrastructure and car-related measures like congestion charges or increased parking fees. The results of the current case study in the Munich area also align with another study [305] that showed that frequent public transit services (hence, public transit quality) is essential in such integrated systems in order to be efficient for passengers.

6

UNCERTAINTIES OF DOWNSCALED SCENARIOS

Truth is not that which is demonstrable but that which is ineluctable.
— Antoine de Saint-Exupery

The chapter is based on contributions from the following publications:

Saprykin, A., Chokani, N. & Abhari, R. S. Uncertainties of sub-scaled supply and demand in agent-based mobility simulations with queuing traffic model. *Networks and Spatial Economics* **21**, 261 (2021)

Saprykin, A., Chokani, N. & Abhari, R. S. *Impacts of downscaled inputs on the predicted performance of taxi fleets in agent-based scenarios including Mobility-as-a-Service* in. **201** (Elsevier, 2022), 574

While ABMs can be used to model the behaviour and complex interactions of people together with urban infrastructure, large-scale and detailed agent-based models also pose a high computational burden. One way to improve the runtimes of such scenarios is to improve the simulation process itself, through more efficient utilization of hardware resources, through better algorithms, parallelization and distributed computing, and the use of specialized hardware accelerators like GPUs and FPGAs. Another approach adopted by researchers is to downscale simulation inputs, namely supply and demand, and scale up the results afterwards. This method allows scenarios of smaller scales to be run faster using aggregated behaviour and without changing the simulation process, as well as avoiding the need for expensive hardware.

When using the latter method, each agent in a simulation represents an aggregate group of modelled individuals. It is evident that such aggregated behaviour in a simulated scenario can lead to errors. For example, a scaled road network introduces rounding errors (such as, when the physical length of a street is too short when large downscaling factors are used), or there may be an over- and under-supply of travel demand on links and for public transit due to the skew of the sampled population on certain routes and at certain locations. Depending on the context of the scenario, distortions in the simulation outcomes may lead to a biased interpretation of the results. In light of the ongoing shift to e-mobility, one can also consider the impact on electricity distribution grids: in a simulation with a downscaled population, the demand for charging would be concentrated at certain aggregated points and may lead to the overloading of power lines, while in reality the demand may be distributed more uniformly in space and time without causing line overloading issues. Another open question is how one should properly scale the charging infrastructure of BEVs in a simulation with a downscaled population.

It should be noted that in general ABM introduces uncertainties into simulations [312, 313] because input data, such as synthetic population and travel demand, are typically derived from surveys and aggregated statistics. Furthermore, the modelled patterns of behaviour are also mostly based on surveys. In contrast to the almost unavoidable uncertainties that result from input data, the uncertainties that result from the use of downscaled input data are introduced artificially because of the lack of required computing power, and these uncertainties can be avoided either by employing a more efficient simulation approach or by running the simulations on more powerful and expensive hardware. Until now, there has been relatively little effort made to understand and quantify the uncertainties introduced by sampling the demand and supply in mobility simulations. Moreover, there is strong evidence (see below) that, in comparison to a simulation with the full-scale input data, sampling distorts the spatio-temporal characteristics of the simulated travel demand and of the traffic externalities. Moreover, while the impacts of downscaled inputs on simulations of car traffic and public transit systems have received some limited attention, the impacts on simulations of coordinated taxi fleets have not been addressed.

This chapter quantifies the impacts of using downscaled populations and downscaled inputs in agent-based scenarios that use a mesoscopic queueing model for traffic propagation. The main question tackled here is the following: what does it mean, in terms of output errors, to run a downscaled agent-based mobility scenario? While GEMSim pushes forward the capabilities to run large-scale agent-based mobility scenarios, it remained unclear if one actually needed the full-scale model to be run, and this chapter closes that gap.

6.1 BACKGROUND

There are only a few multi-agent mobility simulators capable of running truly large-scale and multi-modal scenarios with millions of agents and millions of links and nodes in the network. They include TRANSIMS [101], MATSim [76], and GEMSim. MATSim, as one of the most widely used, provides the main evidence for issues with downscaled populations, and mostly for cars and public transit.

It is common among researchers to use small population samples (1% to 10%) to run large-scale scenarios with scaled input data. Hülsmann et al. [314] used 1% of the population to run a Munich scenario while studying traffic-related air pollution. The speed of computations was the major reason for downscaling, and the simulation output results were scaled back up to 100% of the population. Zhang et al. [315] used 1% of the population to run a large-scale scenario for the city of Shanghai. The memory constraints of the available hardware were stated as the main reason for downscaling. Bekhor et al. [316] integrated activity-based and agent-based models for the Tel Aviv metropolitan area using 10% of population to avoid long running times. Kickhofer et al. [317] used a population sample of 0.65% to run a large-scale scenario for the city of Santiago de Chile. The authors

concluded that the congestion patterns of the simulated scenario do not match the real patterns well and recommended expanding the population sample in the range of 10% to 100%.

More scenarios and case studies with population samples are available in the MATSim book [76]. Furthermore, MATSim developers present in the aforementioned book the use of a 1% to 10% samples of the population to obtain reliable results with the acceptable runtimes. Therefore, the most popular sample sizes for large-scale scenarios run with MATSim are within this range.

In recent years, ABM simulations using public transit and other emerging modes of transport have increased. As a consequence, the required computing power has also increased, and researchers have continued to use small population samples so that the runtimes of their simulations are kept reasonable. However, there is evidence that small population samples are the source of some discrepancies in traffic simulations.

Ben-Dor et al. [108] faced issues using network links shared by private cars and public transit vehicles in MATSim simulations of the Tel Aviv metropolitan area with a 10% population sample. In the simulations, buses tended to get stuck in long waiting queues because the link flow capacity was over-used by cars, resulting in disruptions to public transit services. The study showed that when a coarse population sample is used, the traffic flows on links are not scaled by the same ratio. Furthermore, while for private cars it is possible to calibrate the predicted road traffic flows in order to reduce the adverse effects of scaling, public transit has to run according to schedule, making it more sensitive to changes in network capacity.

Bischoff and Maciejewski [109], in a study of autonomous taxis in the Berlin area, showed that use of a 10% population sample leads to 11% of the demand for autonomous taxis compared to a simulation of the full population; that is a 10% relative error. On the other hand, changes in fleet occupancy statistics were considered acceptable, and the deviations in the durations of both pickup trips and trips with customers were no more than 3% of their length.

Simoni et al. [110] compared the accumulation-production relationship for the links in a MATSim simulation of central Zurich using different sized population samples (10%, 20% and 50%). The simulation results were seen to vary with the size of the population sample. It was also noted that flows on links decreased faster with the increased flow density of the larger population samples. However, only qualitative graphical comparisons were presented, without any quantitative results.

Erath et al. [111] had artefacts of overcrowded buses in a MATSim simulation of public transit in Singapore using a 10% population sample. In order to improve the simulation, a 25% population sample was used. No further details on the issues with public transit were presented.

Bösch et al. [112] in the Switzerland baseline scenario for MATSim drew attention to the use of population downscaling by providing an example from car-sharing simulations: using a smaller population sample means that a reduction in shared vehicles is required to prevent over-supply, and at the same time reduction of the number of shared vehicles leads to reduced availability in the area or under-

supply. Thus, it is not recommended to use population samples of less than 5%–10% for scenarios with shared cars. Issues with the scaling of public transit were also mentioned: because of the inability to scale a public transit schedule itself (that is, a fleet size, the frequency of operation), it is suggested to scale down the size (and, as consequence, link flow capacity consumed by a vehicle) of public transit vehicles proportionally to the size of the population sample. But as shown earlier [108], this approach does not solve the issues with public transit completely when using small population samples.

Kwak et al. [113] conducted probably the first attempt to systematically study the errors that arise in traffic simulations due to the use of samples of the full population. In their study, a macroscale static traffic assignment model was used for different periods of the day. This traffic model was not an agent-based model, but rather used OD matrices with zones. The OD matrices were scaled up to match the flows of the full population. While not an ABM simulation, the work nevertheless showed that the use of samples of the full population affects the predicted traffic flows even at the macro-scale.

Llorca and Moeckel [114] studied the effects of downscaled populations in agent-based traffic simulations of the Munich metropolitan area. The study focused mostly on the impacts on average travel time and the distribution of travel times. The results showed that the average travel time depends on the size of the population sample and is minimal with a sample that is 10%–20% of the full population. Travel time distributions for 5% and 100% of population samples were observed to be very similar. Another interesting finding was that the scale factor for spatial length of the links (streets) did not have a strong influence on the average travel time of agents. The authors concluded that a scale factor of 5% seems reasonable for simulations where only analysis of highly aggregated results is required. However, the authors emphasized that for traffic flow analysis of a single corridor a full population sample is likely required.

While there are a few other studies [115, 116] focused on the impacts of downscaled inputs in relation to car traffic, the scale of their scenarios is relatively small, and the results are subjective. These works suggest that for analysis of disaggregated performance metrics, population samples of at least 25%–30% are used, while small populations samples of 5%–10% can only be used when analysis of highly aggregated simulation results is required.

6.2 SIMILARITY MEASURE

In order to evaluate the impacts of using downscaled input data, a similarity measure was required. This similarity measure must not only be able to compare the outputs from two simulations in relation to each other, but must also provide a quantitative measure of how close the predictions from a simulation with downscaled input data are to a scenario with the full-scale population. As the goal was also to compare simulation outputs both spatially and temporally, a spatio-temporal point was

utilized for the generalization of a similarity measure. A spatio-temporal point represents a measure taken at a certain location over a certain period of time. For traffic flow, the measure may be a set of counts from inductive loops throughout a day, and for public transit the measure may be vehicle occupancy at each stop along a route.

As the problem of uncertainty quantification of downscaled populations has received little attention in prior studies, there are no specific measures suggested in the literature. However, one can see that a comparison of the outputs of two simulations is very close to the procedures used in the calibration and validation of traffic models. While these procedures compare outputs from simulations using a common set of measures of goodness-of-fit, most of these measures have a notable limitation: one must specify a threshold that indicates whether the goodness-of-fit is considered to be acceptable or not.

A brief description of the most commonly used measures of goodness-of-fit, with their limitations and applicability to the present study, is provided below. Some of the measures were subsequently evaluated to see if they qualitatively captured the same trends in goodness-of-fit as identified in the proposed new measure described later.

The mean absolute error (MAE), mean absolute normalized error ($MANE$), root mean squared error ($RMSE$) and root mean squared normalized error ($RMSNE$) are used by many researchers for calibration purposes [318–320]. MAE and $MANE$ are insensitive to large errors, and the non-normalized $RMSE$ may give a biased assessment as different roads have different traffic volumes throughout a day; nevertheless, the $RMSNE$ is a good candidate for further evaluation:

$$RMSNE = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} \left(\frac{x_i - y_i}{y_i} \right)^2} \tag{6.1}$$

where x_i is the prediction at the i-th spatio-temporal point, y_i is an empirical or observed value at the same point, and N is the total number of evaluated spatio-temporal points.

While the $RMSNE$ can show system-wide relative differences in the outputs of scenarios, the $RMSNE$ does not show the nature of the differences. For that, Theil's inequality coefficient [321] is a preferred measure and has been used in numerous studies [319, 322]:

$$U = \frac{\sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2}}{\sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i)^2} + \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i)^2}} \tag{6.2}$$

An inequality coefficient of 0 (zero) indicates a perfect fit, while an inequality coefficient of 1 (one) indicates the worst possible fit. Theil's inequality coefficient can be decomposed into three parts:

$$U_m = \frac{N(\overline{y} - \overline{x})^2}{\sum_{i=1}^{N} (y_i - x_i)^2} \tag{6.3}$$

$$U_s = \frac{N(\sigma_y - \sigma_x)^2}{\sum_{i=1}^{N}(y_i - x_i)^2} \tag{6.4}$$

$$U_c = \frac{2 \cdot (1 - r) \cdot N \cdot \sigma_x \cdot \sigma_y}{\sum_{i=1}^{N}(y_i - x_i)^2} \tag{6.5}$$

where $\overline{x}$ and $\overline{y}$ are the averages of the predicted and the observed quantities X and Y, σ_x and σ_y are standard deviations of the quantities, and r is a Pearson's correlation coefficient of the quantities. The three parts, the bias proportion U_m, the variance proportion U_s, and the covariance proportion U_c indicate the sources of the differences: respectively, the systematic bias, the distribution mismatch between the predicted and the observed quantities, and the non-systematic bias. A value of 0 (zero) for U_m and U_s and a value of 1 (one) for U_c indicate a perfect match between the predicted and the observed quantities. Additionally, the sum of the three parts of Theil's inequality coefficient is unity:

$$U_m + U_s + U_c = 1 \tag{6.6}$$

While the $RMSNE$ and Theil's coefficient are generally applicable to any type of simulation model, the GEH measure of goodness-of-fit has been adopted by many highway agencies around the world (USA [323], UK [324], Australia [325], New Zealand [326], etc.) for the validation of traffic models:

$$GEH(x_i, y_i) = \sqrt{\frac{2 \cdot (x_i - y_i)^2}{x_i + y_i}} \tag{6.7}$$

$GEH(x_i, y_i)$ values less than 5 are considered to be indicative of a good fit; values in the range of 5 to 10 indicate that the model's outcomes are still acceptable but some inconsistency in the model's predictions is present; and values greater than 10 require that the inconsistencies be explained in order for the model's outcomes to be accepted. Typically, at least 85% of spatio-temporal points must yield $GEH(x_i, y_i)$ values less than 5 in order for a model to be considered properly validated. Although $GEH(x_i, y_i)$ is not a real statistic, it is very similar to the chi-squared two-sample test statistic introduced by Pearson, which is for a pair of measurements x_i and y_i defined as:

$$\chi^2(x_i, y_i) = \frac{(x_i - y_i)^2}{x_i + y_i} \tag{6.8}$$

where $\chi^2(x_i, y_i)$ approaches a χ^2 distribution with one degree of freedom. The tested null hypothesis H_0 is that x_i and y_i come from the same distribution, and H_0 is accepted if the value of the $\chi^2(x_i, y_i)$ test statistic is less than the critical value of $\chi^2_{1,\alpha}$ distribution for a chosen significance level α (typically 0.05) and one degree of freedom; otherwise, the null hypothesis H_0 is rejected. As predicted outputs that are inherently variable must be compared, it seems reasonable to use the GEH measure of goodness-of-fit to assess similarity with a selected significance level. However, even though GEH has been adopted by many highway agencies for the validation

of predictions compared to field data, as discussed below, GEH is not considered to be a good similarity measure for two predicted outputs.

From Equation 6.7 and Equation 6.8 one can obtain:

$$\chi^2_{GEH}(x_i, y_i) = \frac{GEH(x_i, y_i)^2}{2} \tag{6.9}$$

where $\chi^2_{GEH}(x_i, y_i)$ approaches a χ^2 distribution with one degree of freedom. It can be shown that a given critical value GEH_{th} used to test $GEH(x_i, y_i)$, with its corresponding value of $\chi^2_{GEH,th}$, acts as a non-linear scaling factor for the critical value of $\chi^2_{1,\alpha}$ that is used to test the chi-squared statistic for the same pair of quantities x_i and y_i:

$$K_{GEH} = \frac{\chi^2_{GEH,th}}{\chi^2_{1,\alpha}} = \frac{GEH^2_{th}}{2 \cdot \chi^2_{1,\alpha}} \tag{6.10}$$

In other words, the use of the GEH formula is similar to using the chi-squared test statistic, but with the critical value of $\chi^2_{1,\alpha}$ corresponding to the different and shifted significance level α. K_{GEH} establishes the relation between these two different critical values of the χ^2_1 distribution. Essentially, the value of $K_{GEH} > 1$ decreases (and shifts) the initial significance level α (the rate of type I errors) in the chi-squared test, therefore increasing the rate of type II errors significantly. Moreover, the same initial (and unshifted) value of α is used to make the decision on the goodness-of-fit for the model that is being validated: the number of spatio-temporal points that needs to satisfy the scaled critical value of $\chi^2_{1,\alpha}$ is not increased due to the decreased α. It is for this reason that GEH is not considered to be a real statistic.

For example, the recommended value of $GEH = 5$ with an initial critical value of $\chi^2_{1,\alpha} = 2.07$ for a significance level $\alpha = 0.15$ (that is, the requirement that at least 85% of the spatio-temporal points pass the GEH test) gives approximately $K_{GEH} = 6$, thus shifting α to 0.00041 while ensuring that 85% of points pass the GEH test. Similarly, a threshold value of $GEH = 4$ with at least 95% of the points passing the GEH test (that is, initial $\alpha = 0.05$), gives $K_{GEH} = 2$ and a shifted $\alpha = 0.00468$.

Another interesting interpretation of the GEH formula can be obtained from the following:

$$\chi^2(x_i, y_i) = \frac{(x_i - y_i)^2}{2 \cdot E_{xy,i}} = \frac{(x_i - E_{xy,i})^2}{E_{xy,i}} + \frac{(y_i - E_{xy,i})^2}{E_{xy,i}} = \frac{2 \cdot \sigma^2_{xy,i}}{E_{xy,i}} \tag{6.11}$$

where $E_{xy,i}$ and $\sigma^2_{xy,i}$ are average and variance of x_i and y_i. The variance $\sigma^2_{xy,i}$ is multiplied by a factor of 2 because both x_i and y_i are random variables. Combining Equation 6.10 and Equation 6.11 gives:

$$K_{GEH} = \frac{\chi^2_{GEH,th}}{\chi^2_{1,\alpha}} = \left. \frac{\sigma^2_{GEH,th} \cdot E_{1,\alpha}}{E_{GEH,th} \cdot \sigma^2_{1,\alpha}} \right|_{E_{1,\alpha} = E_{GEH,th}} = \frac{\sigma^2_{GEH,th}}{\sigma^2_{1,\alpha}} \tag{6.12}$$

where $\sigma^2_{GEH,th}$ and $\sigma^2_{1,\alpha}$ are variances for the corresponding critical values of $\chi^2_{GEH,th}$ and $\chi^2_{1,\alpha}$ conditional on the equality of corresponding means $E_{GEH,th}$ and $E_{1,\alpha}$.

Equation 6.12 shows that GEH with $K_{GEH} > 1$ yields model predictions with a higher variance than allowed with the chi-squared statistic, and over-dispersion is exactly defined by K_{GEH}.

The lack of clarity in how the GEH_{th} values that are suggested as critical for traffic model validation are determined casts doubt on the general applicability of the GEH formula as a good similarity measure for the outcomes from simulations. Neither of these GEH_{th} values can be considered to be a reliable statistical measure. Instead, the chi-squared test statistic is considered to be more suitable for assessing similarity. This statistic is perhaps the most widely used statistic, with well-known properties. Moreover, the chi-squared test allows one to choose a significance level α, making this measure of goodness-of-fit more widely applicable. Nevertheless, it worthwhile to see if the GEH measure can also capture the trends in similarity.

Considering the above, the following similarity measure for a given significance level α was proposed:

$$S_\alpha = min\{\frac{\sum_{i=1}^{N} \delta_i}{N \cdot (1-\alpha)}, 1\} \tag{6.13}$$

where

$$\delta_i = \begin{cases} 1, & \text{if } \chi^2(x_i, y_i) < \chi^2_{1,\alpha} \\ 0, & \text{otherwise.} \end{cases} \tag{6.14}$$

S_α has a minimum of 0 (zero) when none of spatio-temporal points pass the chi-squared test, and a maximum of 1 (one) when at least $(1-\alpha) \cdot 100\%$ of spatio-temporal points pass the test. Thus, S_α represents a straightforward statistical and quantitative approach to measure the similarity of outputs from mobility simulations. This similarity measure is generic enough that it can be applied to both traffic flows and occupancy of public transit vehicles.

The following four measures are further evaluated in subsequent sub-sections: the $RMSNE$, Theil's inequality coefficient, GEH and the proposed S_α.

6.3 DOWNSCALING SCENARIOS

6.3.1 *Cars and public transit*

The older Switzerland scenario, described in Section 2.4, was used to access the impact of scenario downscaling on car traffic and public transit occupancies. To scale the network for the size of the population sample, coefficients k_l and k_f of the queueing buffers in Equations 2.16–2.17 were used. Typically, when full scale input data is used, the coefficients are equal or close to one, while for downscaled input data the coefficients are equal or close to the fraction of the population sample that is used for sampling. The reason why these coefficients do not necessarily scale in direct proportion to the size of a sample is that the simulation process is highly non-linear and stochastic. The dependence of these coefficients on the scale that is used is evaluated later in the chapter.

To investigate the impact of scaling one's input data, the following set of population samples was used: 1%, 2%, 5%, and 10%–90% at 10% intervals. Uniformly random sampling of the full population was applied, and the simulation outputs with the full population (that is, 100% population sample) were used as the reference (that is, quantity Y). Hence, all simulated population samples were compared to the full population sample simulated as a reference case. A 30-hour period, starting from midnight, was simulated. As the dependence of the network parameters k_l and k_f on the size of population sample is unknown, a calibration (impedance matching) was performed for each of the samples to match the simulation outputs of the reference case as closely as possible. Two sets of spatio-temporal points were chosen for the calibration of car traffic: counts on all car-related network links during the morning (07:00–08:00) and the evening (17:00–18:00) peak hours. During simulation, the traffic counts are aggregated over 15-minute time intervals, and therefore for each peak hour each network link has four distinct spatio-temporal points that were used in the assessment of similarity. The use of multiple spatio-temporal points on each link allows the temporal dynamics of traffic flows to be compared rather than limiting the comparison to only time-averaged states. In addition to the peak hours, two other sets of spatio-temporal points were used to evaluate similarity measures with calibrated coefficients: noon hour (12:00–13:00) when the traffic is relaxed and daily (00:00–24:00) aggregated values. Network links where no cars pass in the reference case were excluded from the analysis. As public transit was not scaled (neither its network links nor the schedule), it need not be calibrated. The occupancy of each public transit vehicle after its departure from each stop were used as the set of spatio-temporal points to assess the similarity of simulated public transit. The traffic and occupancy counts from simulations with population samples were scaled up by the inverse of the sampling fraction size to match the counts from the reference case.

The determination of the optimal values k_l^* and k_f^* for the calibrated parameters k_l and k_f was the subject of the following optimization problem:

$$\begin{aligned} \max \quad & S_\alpha^m(k_l, k_f) + S_\alpha^e(k_l, k_f) \\ \text{s.t.} \quad & \{k_l, k_f\} \in (0; 1] \end{aligned} \tag{6.15}$$

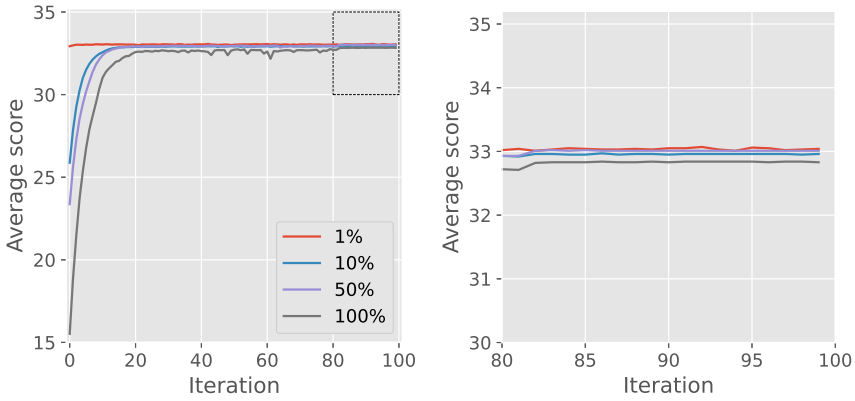
where a significance level α of 0.05 was chosen, and S_α^m and S_α^e are the similarity measures for the morning and evening peak hours, respectively. To reduce the number of simulations required for the calibration, the search space for values k_l^* and k_f^* was adjusted depending on the sampling fraction size as follows:

- $\{k_l, k_f\} \in [0.1; 1]$ with a step of 0.1 for population samples from 10% to 90%;
- $\{k_l, k_f\} \in [0.005; 0.015]$ with a step of 0.001 for the 1% population sample;
- $\{k_l, k_f\} \in [0.01; 0.1]$ with a step of 0.01 for population samples of 2% and 5%.

Additionally, the small population samples (less than 10%) were examined with a broader range of k_l and k_f, but the variation in the optimization objective was in

the range of 1%. Similarly, larger population samples did not show significantly different behaviour for intermediate scaling ratios. In total, 1 221 simulations were performed for the whole set of population samples. After calibration, the k_l^* and k_f^* determined for each population sample were used to calculate similarity measures of car traffic for the population sample. This procedure was repeated five times with different sets of the random population samples and the results of the similarity measures were averaged to smooth out fluctuations. Thus, a total of 6 105 simulation runs were performed.

To determine the number of iterations to run for each of the population samples, the average score of the agents between iterations was examined for different sample sizes. Figure 6.1 (left side) shows that larger population samples require more iterations to converge the average score, while samples smaller than 10% require only a few iterations. The same behaviour, when the number of iterations required for a simulation to converge depends on the population sample size, was previously reported by [114]. Moreover, using larger population samples may lead to oscillations of the score between iterations. Figure 6.1 (right side) shows a zoomed-in part of the previous plot after 80 iterations, when agents stopped innovating their daily plans and only chose one of the previously memorized daily plans with the best scores. After 100 iterations, the difference of the average score between 1% and 100% samples is less than 1%. However, after 20 iterations, there is only marginal improvement for the average score even for large samples. Hence, downscaled scenarios were run for 20 iterations with a re-routing strategy followed by 5 iterations with up to three previously memorized plans only.



(a) Overview of the whole simulation.

(b) Zoomed-in part (dashed).

FIGURE 6.1: Average score of agents depending on the population sample size during the simulation of Switzerland, used to identify the minimum number of iterations to run before evaluating the uncertainties of downscaled scenarios.

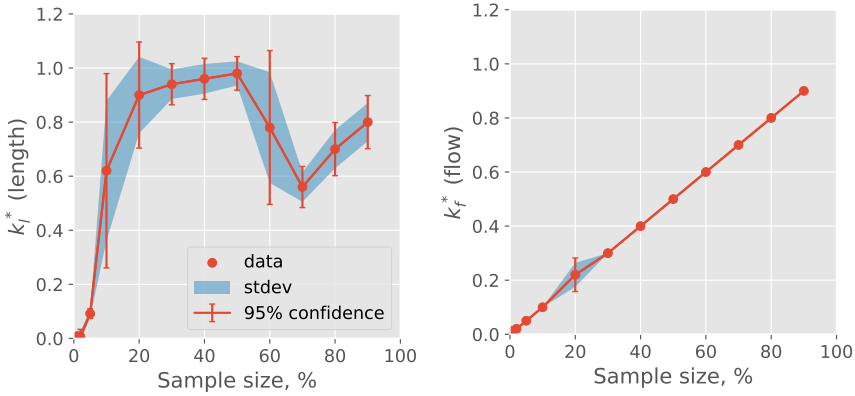
6.3.2 *Coordinated fleets*

The Munich scenario from Chapter 5 was used to evaluate the impacts of downscaled scenarios on predicted fleet performance. The simulations were conducted as follows. First, a simulation was run for 350 iterations, with 10% of the agents re-routed after each iteration, to converge to an equilibrium. Then, in successive simulations, a new transport mode, MaaS, was made available to the agents; the MaaS includes a combination of public transit and taxi rides for the first and last mile travel legs. The taxi trips were limited to a direct distance of up to 3 km, and the total trip length must be of at least 6 km to prevent the use of taxis for uni-modal travelling. The area of fleet operation is shown in red in the left plot of Figure 5.1, and only agents who travel within this area could switch to the MaaS mode. The fleet size was set to 15 000 vehicles. The simulation was run again for 700 iterations: first, 350 iterations with 10% of the agents re-routed and 10% of the agents allowed to switch the mode between either car, public transit, or MaaS; and then 350 iterations where agents can pick only one of the previously experienced plans. In the end, about 7.5%, or 252 389, of the agents switched to the MaaS mode. This converged reference scenario was used to assess the impacts of downscaling, which was performed by sampling, uniformly at random, a fraction of agents from the reference scenario and adjusting the flow capacities of the road network according to optimal coefficients obtained for the Switzerland scenario. The fleet size was downscaled in proportion to the size of the population sample. Each downscaled scenario was run for 150 iterations to converge: 100 iterations with 10% re-routing and 50 iterations using plans from the agents' memories. Sample sizes of 1%, 2%, 5%, and 10%–90% in increments of 10% were used. The memory of agents was always set for the five last chosen plans.

6.4 RESULTS

6.4.1 *Cars and public transit*

The mean, standard deviation (shaded area) and 95% confidence intervals using t-distribution (vertical bars) for the optimal values of the scaling coefficients k_l^* and k_f^* are shown in Figure 6.2. The standard deviation and confidence intervals were obtained based on five runs for each of the combinations of k_l and k_f during the calibration process as described above. It is interesting to note that the capacity coefficient k_f^* matches exactly the scale of the input data for almost all population samples; thus, it is evident that flow capacity is the main driver of the traffic dynamics in the traffic model. For almost all population samples, the standard deviation of k_f^* is either zero or very close to zero; therefore the flow capacity coefficient is stable and is not affected by traffic fluctuations between simulation runs.



(a) Spatial scaling coefficient (k_l^*).

(b) Capacity scaling coefficient (k_f^*).

FIGURE 6.2: Optimized scaling coefficients for spatial and capacity buffers of the traffic queueing model for the road network of Switzerland.

The spatial coefficient k_l^* is close to one for population samples in the range of 20%–50%, and decreases for larger-sized population samples. However, for very small population samples, less than 10%, k_l^* decreases substantially, and does not match the overall input scale factor. This can be explained as follows: the more the network is downscaled, the more the performance of spatial buffers degrades due to rounding errors when smaller values of k_l are applied. Conversely, larger k_l better matches the impedance of the reference case. The same behaviour is reported by [114] where spatial buffers were scaled with factors larger than the corresponding size of the population sample. The standard deviation of k_l^* is larger than the standard deviation of k_f^*, therefore indicating that spatial buffers are more sensitive to stochastic fluctuations in the simulated traffic.

To show the impact of rounding errors on the simulated traffic flow, consider the following example with a network link of $N_l = 8$ (about 60 meters long) and $N_f = 1$ (flow capacity of 900 cars/hour) which is very typical for the city of Zurich. For the sake of simplicity, one can assume that the flow of this link is fully matched with downstream links such that there is always free space available in downstream spatial buffers. Further, consider a stable flow of 900 cars/hour from upstream links, or 1 car every 4 seconds of simulated time. In a non-scaled scenario, this link should never be overflown, and no spillover should occur under the assumed conditions. In the first scenario, a 40% sample is used, and, according to Equations 2.16 – 2.17, link buffers are scaled to $N_{l,0.4} = 3$ and $N_{f,0.4} = 1$, whereby the link accumulates 0.4 cars of flow capacity every 4 seconds. In the second scenario, a 70% sample is used, and link buffers are scaled to $N_{l,0.7} = 5$ and $N_{f,0.7} = 1$, whereby the link accumulates 0.7 cars of flow capacity every 4 seconds. As soon as the link has accumulated flow capacity $>= 1.0$, it can release 1 car into the capacity buffer and downstream

consequentially. Table 6.1 shows the state of the spatial buffer of the link during a simulation when the upstream flow is kept constant (one car every 4 seconds). Since the flow of a single car cannot be downscaled, there exists a probability that for a certain time period a flow of 900 cars/hour is kept. In Table 6.1, a simulation step equals to 4 seconds to keep the calculations simple.

TABLE 6.1. State of the capacity buffer of a downscaled link from a simulated test case with a constant traffic flow. Queues' first spillover states are in bold.

Step	40% sample		70% sample	
	Flow capacity (0.4/step)	**Queue state ($N_{l,0.4} = 3$)**	**Flow capacity (0.7/step)**	**Queue state ($N_{l,0.7} = 5$)**
1	0.4	1 (0)	0.7	1 (0)
2	0.4 + 0.4 = 0.8	1 + 1 = 2 (0)	0.7 + 0.7 - 1.0 = 0.4	1 - 1 + 1 = 1 (0)
3	0.8 + 0.4 - 1.0 = 0.2	2 - 1 + 1 = 2 (0)	0.4 + 0.7 - 1.0 = 0.1	1 - 1 + 1 = 1 (0)
4	0.2 + 0.4 = 0.6	2 + 1 = 3 (0)	0.1 + 0.7 = 0.8	1 + 1 = 2 (0)
5	0.6 + 0.4 - 1.0 = 0.0	3 - 1 + 1 = 3 (0)	0.8 + 0.7 - 1.0 = 0.5	2 - 1 + 1 = 2 (0)
6	0.0 + 0.4 = 0.4	3 + 1 = **3 (1)**	0.5 + 0.7 - 1.0 = 0.2	2 - 1 + 1 = 2 (0)
...	...	...	...	...
15	0.6 + 0.4 - 1.0 = 0.0	9 - 1 + 1 = 3 (6)	0.8 + 0.7 - 1.0 = 0.5	5 - 1 + 1 = 5 (0)
16	0.0 + 0.4 = 0.4	9 + 1 = 3 (7)	0.5 + 0.7 - 1.0 = 0.2	5 - 1 + 1 = 5 (0)
17	0.4 + 0.4 = 0.8	10 + 1 = 3 (8)	0.2 + 0.7 = 0.9	5 + 1 = **5 (1)**

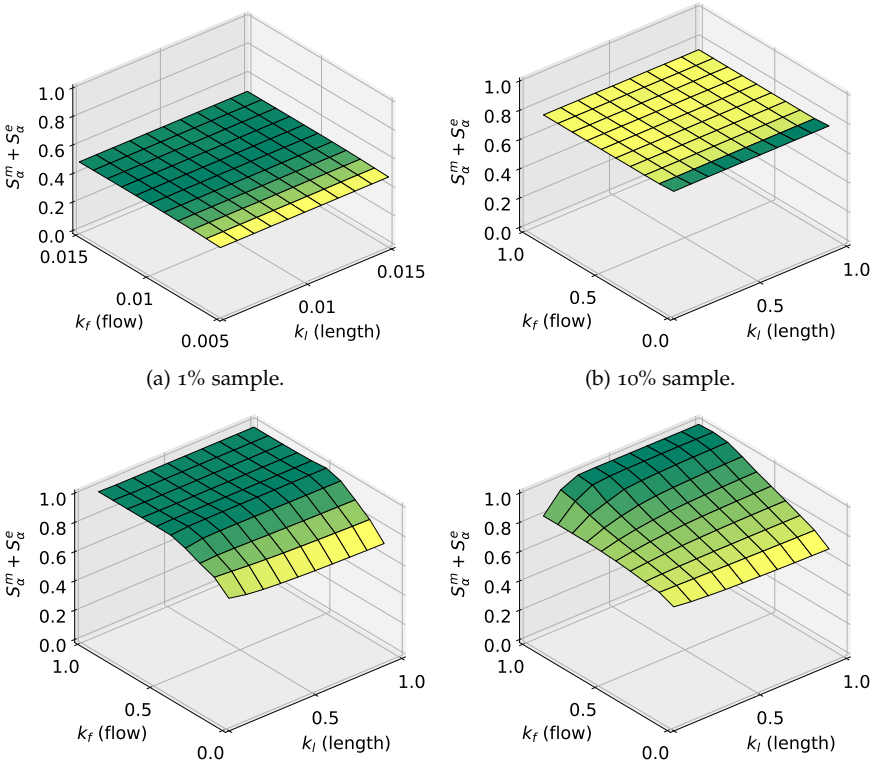
Table 6.1 shows that the spatial buffer of the link is overflown in only 6 steps (24 simulated seconds) when using a 40% population sample with the network downscaled accordingly. However, when using a 70% population sample, it takes 17 steps (68 simulated seconds) to overflow the link. In total, after 17 steps, the link with a smaller scaling factor of 0.4 spills over eight cars causing congestion in upstream links. Hence, rounding errors in downscaled network links can severely affect congestion patterns and traffic dynamics during the simulation.

Additional insight into the nature of the impacts of k_l and k_f is given in Figure 6.3, where the optimization objective for one of the sets of samples (not averaged across five sets) is shown. The characteristics of the optimization objective in the k_l - k_f plane, given in Equation 6.15, are sensitive to the size of the population sample that is used. For very small population samples of 1% the objective is very dissimilar compared to the reference case and the surface of the objective is flat. With this flat surface, there is a high likelihood that many different combinations of k_l and k_f may be optimal. Thus, with the very small population sample, the scaled flow capacity of the network does not drive the traffic dynamics, and the model generates mostly noise rather than proper traffic dynamics. When the size of the population sample is increased to 10%, the model is more similar to the reference case in terms of the traffic dynamics, and the flow capacities of the buffers generate more realistic traffic flows. With a 50% population sample the model is almost fully driven by the flow capacity and generates traffic that is very similar to the reference case. It is also clear that the k_l coefficient does not significantly affect the simulation outputs. With a 90% population sample, the model is sensitive to both the flow capacity and the spatial length of the links; in this regard k_l effectively fine-tunes the traffic dynamics.

As the size of the population sample increases, the objective (traffic similarity) improves, and the surface of the objective in the k_l - k_f plane becomes more concave. The concave surface indicates that the objective is more sensitive to changes of the scaling coefficients, meaning that the optimal values of k_l and k_f are closer to the scaling factor when the surface is concave. Nevertheless, as the simulations are highly non-linear, the optimum coefficient k_l^* for spatial buffers is smaller than the corresponding scale factor for the flow capacity. The optimal values of k_l and k_f together with standard deviations for each of the population samples are given in Table A.1.

The measures of goodness-of-fit for the different population samples are presented in Figures 6.4–6.5. In the legend, GEH_5 and GEH_{10} indicate that the links satisfy, respectively, the conditions $5 \leq GEH(x_i, y_i) < 10$, and $GEH(x_i, y_i) \geq 10$, and HW indicates that the similarity measure was only applied to a subset of network links consisting from motorway and expressway links with a minimum speed of 80 km/h. The GEH measure is given as a percentage of the total number of links that are analysed. A total of 1 033 642 links, including 67 483 motorway and expressway links, were analysed.

As expected, the GEH measure satisfies the validity condition $GEH < 5$ for at least 85% of the links even with the 5% population sample during hourly evaluated periods (morning, afternoon and evening), while the traffic similarity S_α (based on



(a) 1% sample.

(b) 10% sample.

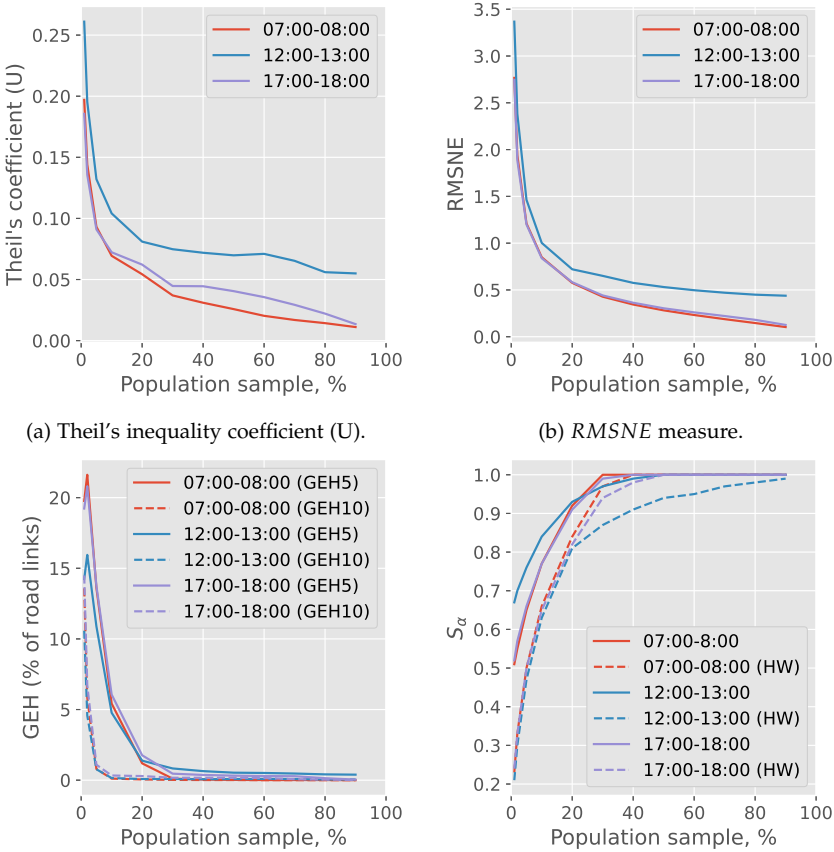
(c) 50% sample.

(d) 90% sample.

FIGURE 6.3: Normalized surfaces of the optimization objective during the calibration procedure of coefficients for the queuing traffic model.

the chi-squared statistic) yields statistically significant results only with a population sample of at least 30% in the morning, 60% in the afternoon and 40% in the evening. The lower similarity in the afternoon and evening peak hours is only observed for motorways and expressways, while for the whole set of links a statistically significant result is already achieved with a population sample of at least 30%. Motorways and expressways are also affected by discrepancies in traffic dynamics when small population samples are used. However, one may not intuitively expect this result considering that links with smaller volumes are affected more by traffic fluctuations, but simulations show that both road types are affected by the size of the population samples. Nevertheless, lower traffic volumes on highways in the afternoon could be the reason for higher fluctuations and lower similarity.

The same trend is also observed for daily aggregated counts in Figure 6.5, where neither GEH nor S_α reaches a statistically significant result, as the error accumulates



(a) Theil's inequality coefficient (U).

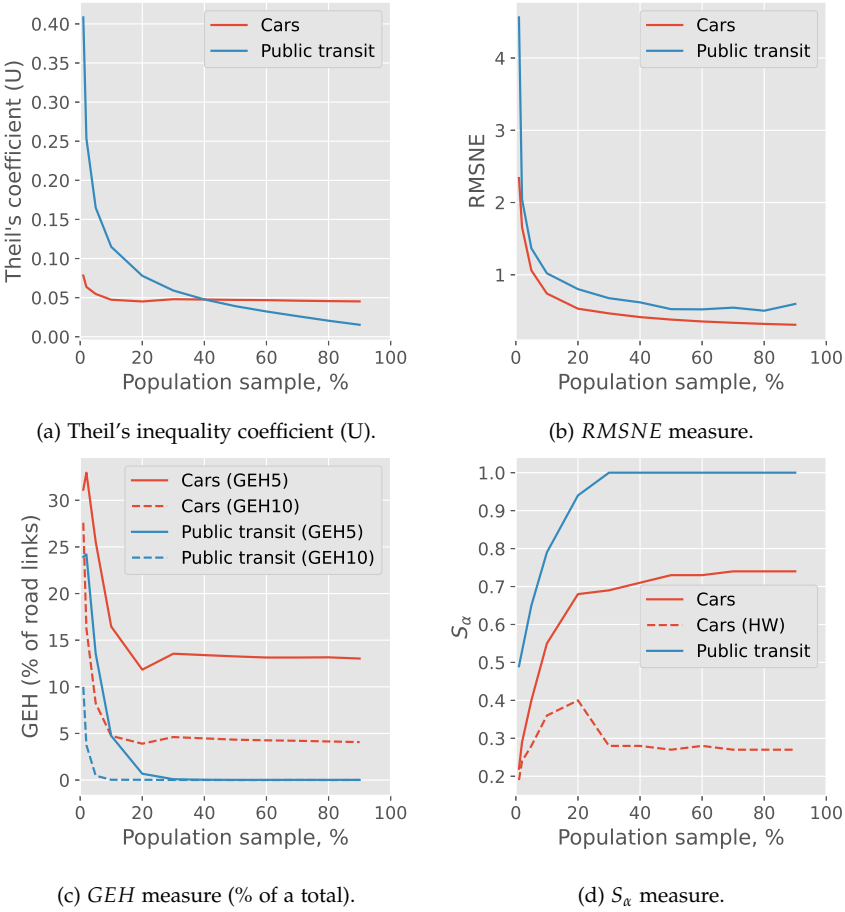
(b) *RMSNE* measure.

(c) *GEH* measure (% of a total).

(d) S_α measure.

FIGURE 6.4: Measures of goodness-of-fit (hourly time periods) for downscaled Switzerland scenarios.

throughout the day for each of the links. It is also noteworthy that as the size of the population sample increases from 1% to 30%, the *GEH* measure decreases sharply from more than 30% to less than 1% (except the daily evaluated period where errors are accumulated), and then decreases only slightly up to a population sample of 90%. Therefore, *GEH* captures qualitatively a critical point with a population sample of 30%, while S_α captures this point quantitatively. The shape of curves for the *GEH* measure with the GEH_{10} condition decreases more rapidly than the curves for the *GEH* measure with the GEH_5 condition as the more heavily penalized links fit better to the reference distribution as the size of the population sample is increased.



(a) Theil's inequality coefficient (U).

(b) *RMSNE* measure.

(c) *GEH* measure (% of a total).

(d) S_α measure.

FIGURE 6.5: Measures of goodness-of-fit (daily time periods) for downscaled Switzerland scenarios.

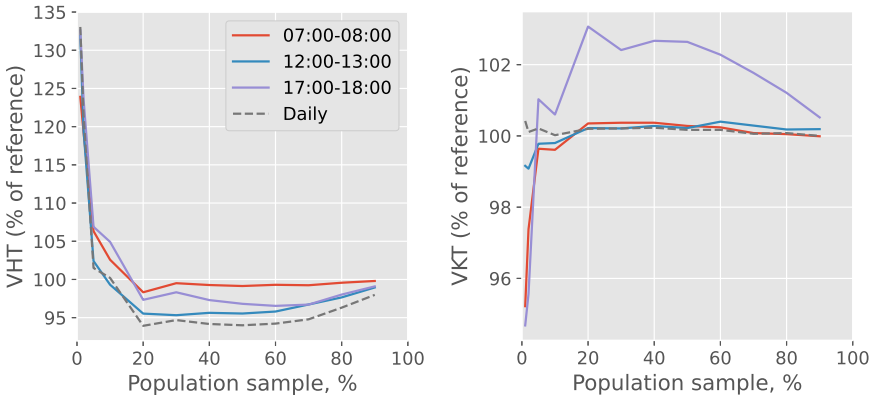
All other measures of goodness-of-fit qualitatively capture the impacts of the size of the population sample on traffic dynamics. However, Theil's inequality coefficient highlights a critical point when the size of the population sample is in the range of 10%–20%; further increases in the size of the population sample lead to only minor improvements. One can also note that for the noon hour the errors are higher, as was previously mentioned. More detailed data on the three component parts of Theil's inequality coefficient are presented in Tables A.2–A.5 for cars and in Table A.6 for public transit. These data show that there is a larger systematic bias and a larger mismatch in the distribution for small population samples. Such errors can be

expected when small traffic volumes are scaled up by large multipliers. For large population samples, non-systematic errors contribute more than the systematic bias and mismatch of distribution, as the overall match of traffic dynamics and transit occupancy is better.

The *RMSNE* also shows that there is a tendency for the error to reduce with larger population samples, but it is very difficult to interpret these results. For example, the *RMSNE* is below 2 for cars when a 5% population sample is used, as the simulation results were averaged over a large number of links when the *RMSNE* were evaluated.

Although the public transit infrastructure was not scaled using k_l and k_f coefficients and public transit vehicles did not share the roads with other cars, in general, all similarity measures indicate that public transit has higher uncertainties in peak hours but the similarity remains almost unaffected in a daily evaluated period. Public transit vehicles always run according to the schedule, and therefore are unaffected by the queueing model (that is, no congestion forms, and vehicles can always move forward). Thus, one can infer that discrepancies in the predicted traffic dynamics that result from the use of small population samples do not derive primarily from the traffic model that is used, but rather are a consequence of the sub-scale population which itself distorts the demand.

Figure 6.6 and Table A.7 show the performance of the simulated car transport in terms of the total VHT and total VKT during hourly and daily evaluated periods; the performance is normalized relative to the reference case. The simulations with a 5% population sample give errors in the range of 5%–6%, and the errors are 2%–3% for simulations with 10%–20% population samples.



(a) Vehicle-hours travelled.

(b) Vehicle-kilometres travelled.

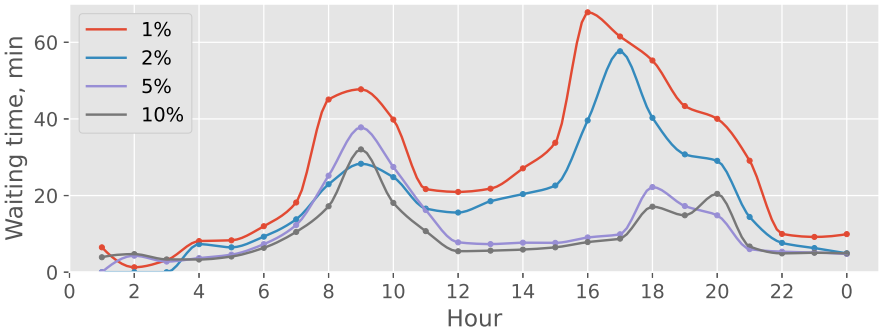
FIGURE 6.6: Performance of the simulated car transport in downscaled Switzerland scenarios.

It is worthwhile to note that VKT varies much less than VHT and has an error of no more than 6%, even with a 1% population sample, while VHT has errors in the range of 7%–30% for small population samples. A possible explanation is that the distribution of trip distances performed by the agents is narrower and more concentrated around the mean value, so it is easier to approximate the distribution with a few agents. On the other hand, the distribution of trip durations is broader, and the use of a small population sample gives a larger error. However, it is also evident that discrepancies in the measures of goodness-of-fit depend on the patterns of behaviour of the simulated population, and these patterns can be different in other scenarios.

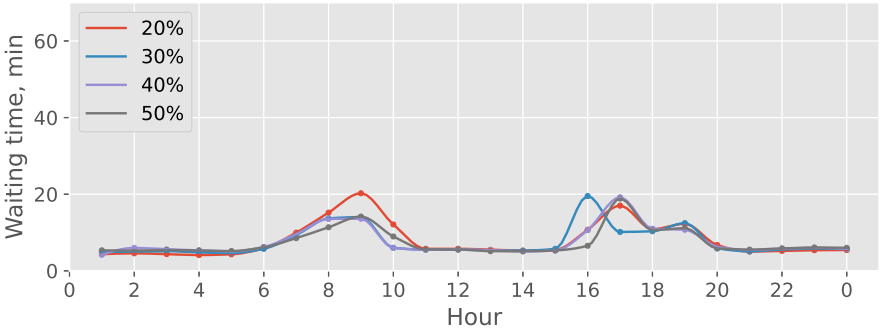
6.4.2 *Coordinated fleets*

The impact of sample size on the average waiting times of clients (the time from when a request is sent to the time when a taxi arrives) is shown in Figure 6.7. For sample sizes less than 5%, the evening peak hour has substantially longer waiting times of about 1 hour compared to about 20 minutes for the larger sample sizes. The waiting times in the morning for smaller sample sizes are also longer and range from about 30 to 50 minutes, compared to the waiting times of 10–20 minutes for sample sizes larger than 10%. The substantially longer waiting times of small sample sizes in the evening peak hour can be explained by the fact that the initial placement of vehicles is based on the known morning-time demand; hence the waiting times are less affected in the morning. The larger sample sizes of 20%–50% yield quite similar results over the whole day, with some differences observed during the evening. As for car traffic, for sample sizes larger than 30% there are no substantial differences in the waiting times. However, for the full-scale population, the longest waiting times occur 3 hours later than with the smaller samples.

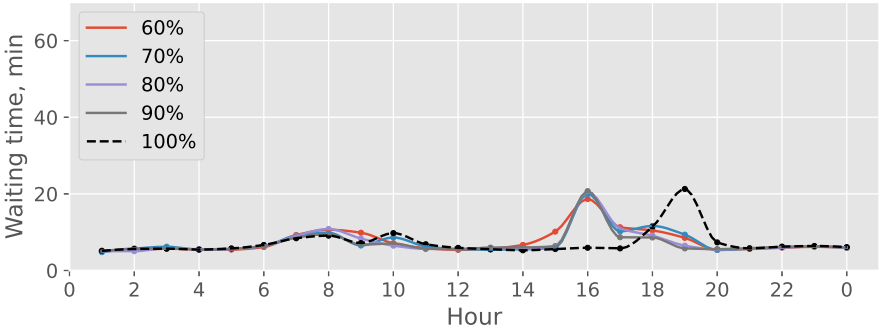
Figure 6.8 compares the impact of sample size on fleet utilization. During the morning peak hour, the fleet utilization differs substantially over the range of 60%–100% for sample sizes of 1% to 10%, while sample sizes of 30% to 50% yield fleet utilization in the range of 70%–80%. Sample sizes larger than 60% consistently yield fleet utilization of around 60% in the morning. All sample sizes have a fleet utilization of 100% in the evening peak hour. However, the distribution of fleet utilization over the evening differs with sample size: for the smaller samples, the distribution tends to be flatter and wider than for the larger samples; only the full-scale population has a narrow distribution with a sharp peak. The flatter and wider distributions of fleet utilization with the smaller samples indicate that for longer portions of the evening peak hours, a fleet operator finds it challenging to schedule the increased number of requests. The most probable reason for this is the spatial imbalance between supply and demand that results when downscaled samples are used; this imbalance is most pronounced in municipalities with low population densities. It is worth noting that the narrow and sharp distribution of



(a) 1%–10% samples.



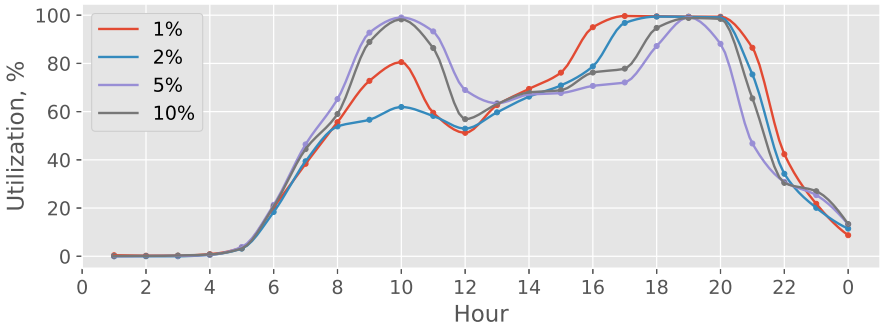
(b) 20%–50% samples.



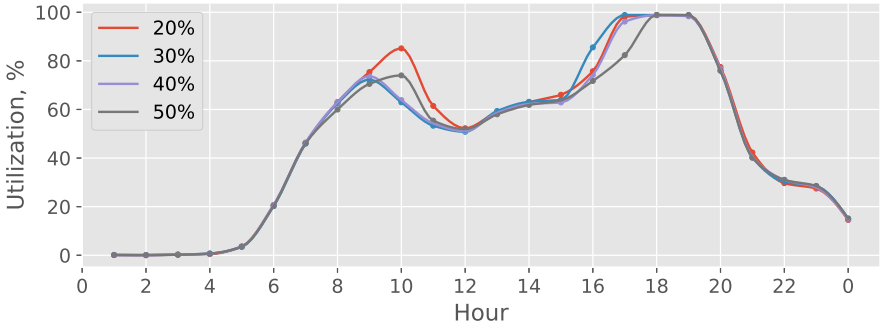
(c) 20%–50% samples.

FIGURE 6.7: Impact of sample size on the average waiting times of agents in downscaled scenarios of the Munich area.

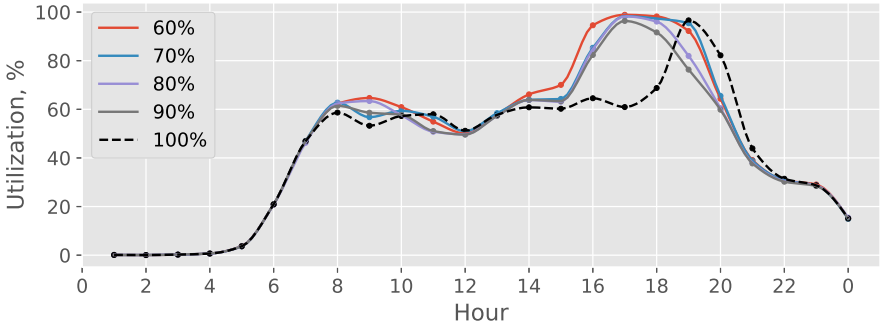
fleet utilization in the full-scale population case explains the shift in the evening peak of waiting times compared to the downscaled sample sizes.



(a) 1%–10% samples.



(b) 20%–50% samples.



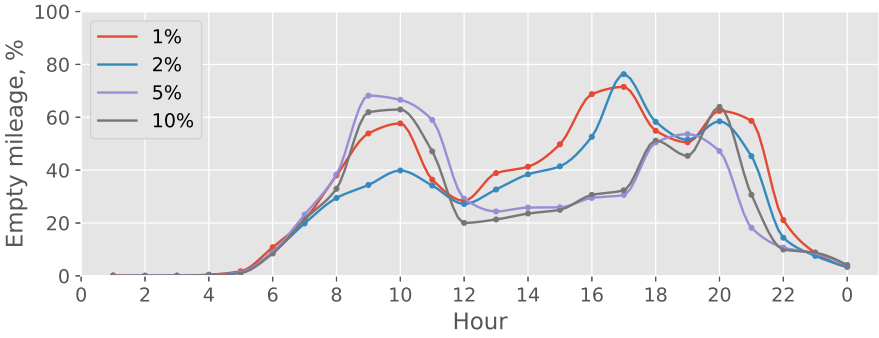
(c) 60%–100% samples.

FIGURE 6.8: Impact of sample size on fleet utilization in downscaled scenarios of the Munich area.

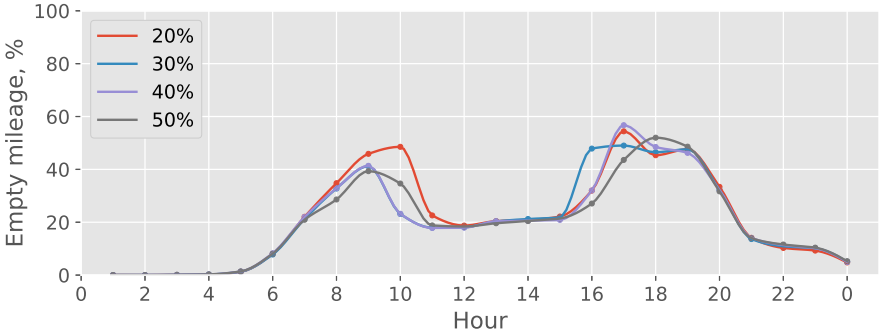
The impact of sample size on the fleet empty mileage is compared in Figure 6.9. It is evident that the fleet tends to drive longer distances with downscaled sample sizes. As with the fleet utilization, for sample sizes of 1% to 10% the fleet empty

mileage differs substantially, especially in the morning peak hour. This is because the smaller samples are more affected by the imbalance between supply and demand. The sample sizes of 30% to 50% differ less across the day, and have a smaller empty mileage in the morning of 40% compared to 60% in the sample sizes of 1% to 10%. The larger samples of 60% to 100% yield empty mileage of 30% in the morning. The magnitudes of the empty mileage in the evening are similar across all sample sizes. However, similar to the fleet utilization, only the full-scale population case has a narrow and sharp distribution of the fleet empty mileage.

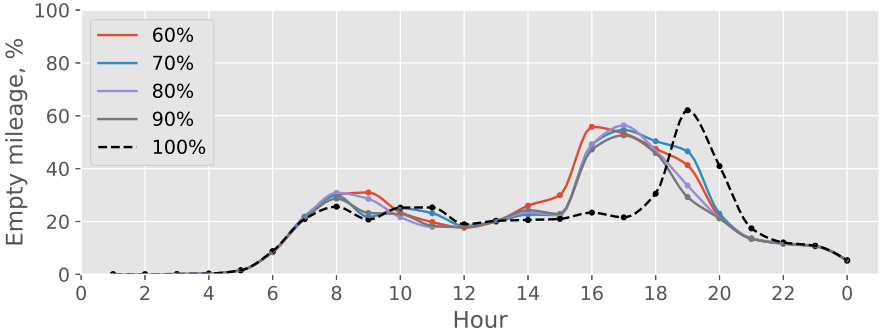
The distributions of the per-vehicle daily driven distances for some sample sizes are compared in Figure 6.10, and the impact of sample size on average per-vehicle daily driven distance is shown in Figure 6.11. It can be seen that the distribution of distances changes with sample size. For sample sizes of 10% and less, the distributions are close to a normal distribution, with mean per-vehicle daily driven distances of 175 km and 250 km for the sample sizes of 1% and 10%, respectively. As the sample size increases, the distributions tend towards a bimodal distribution: for the 100% sample size, the peaks are around 100 km and 270 km, with an overall mean per-vehicle daily driven distance of 215 km. The bimodal distribution can be explained by the fact that the more vehicles and clients are in the system, the higher the probability that a nearby vehicle can be found for each request, and the higher the probability that a vehicle will be idle. This is especially relevant for low-density areas that have fewer vehicles and clients.



(a) 1%–10% samples.

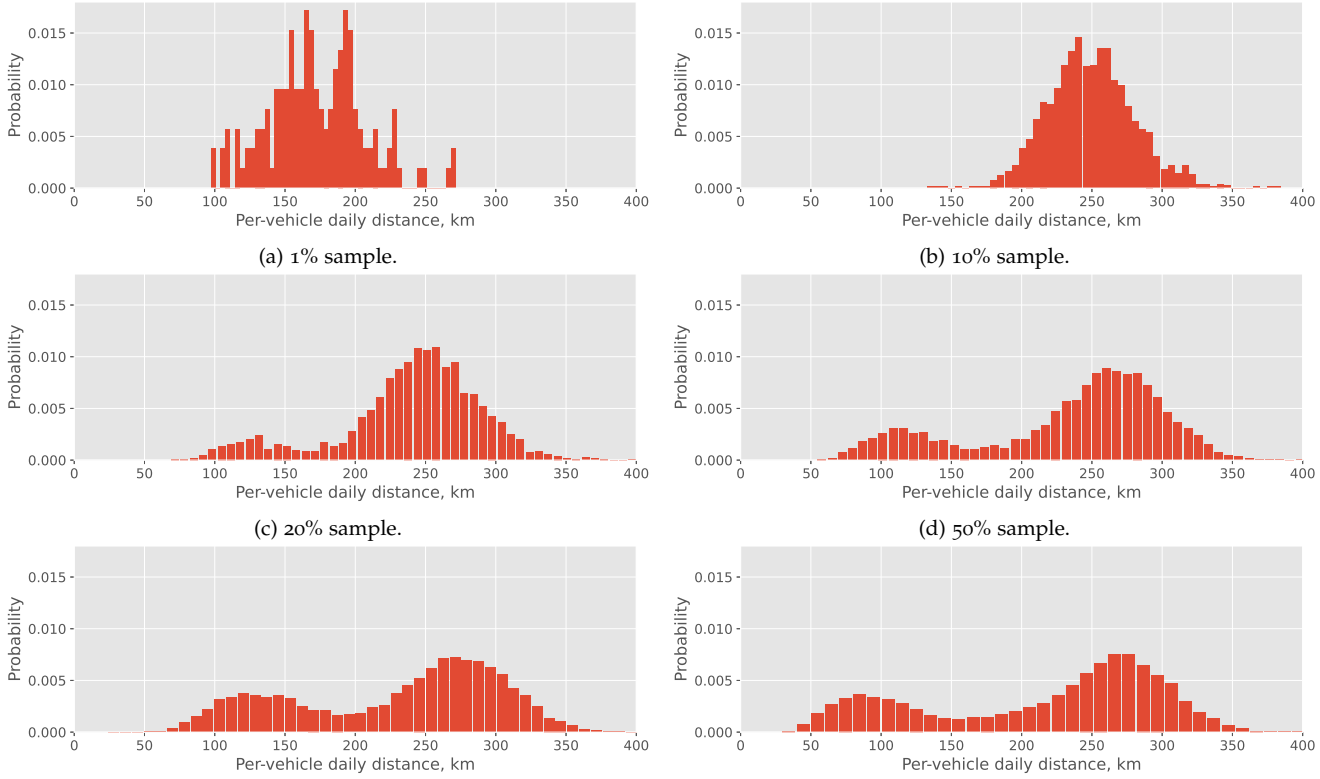


(b) 20%–50% samples.



(c) 60%–100% samples.

FIGURE 6.9: Impact of sample size on fleet empty mileage in downscaled scenarios of the Munich area.



(a) 1% sample.

(b) 10% sample.

(c) 20% sample.

(d) 50% sample.

(e) 80% sample.

(f) 100% sample.

FIGURE 6.10: Distribution of per-vehicle daily driven distance in downscaled scenarios of the Munich area.

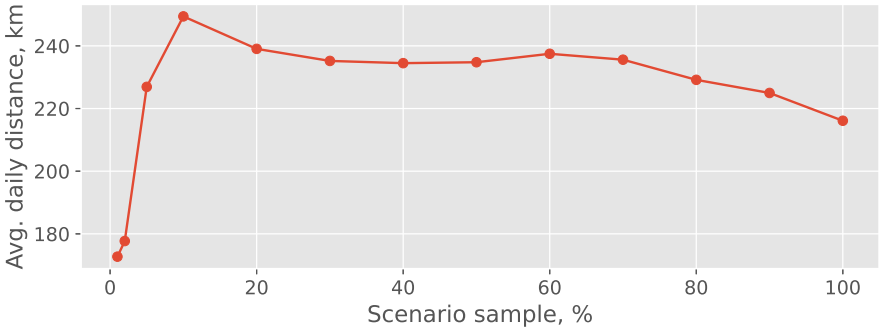
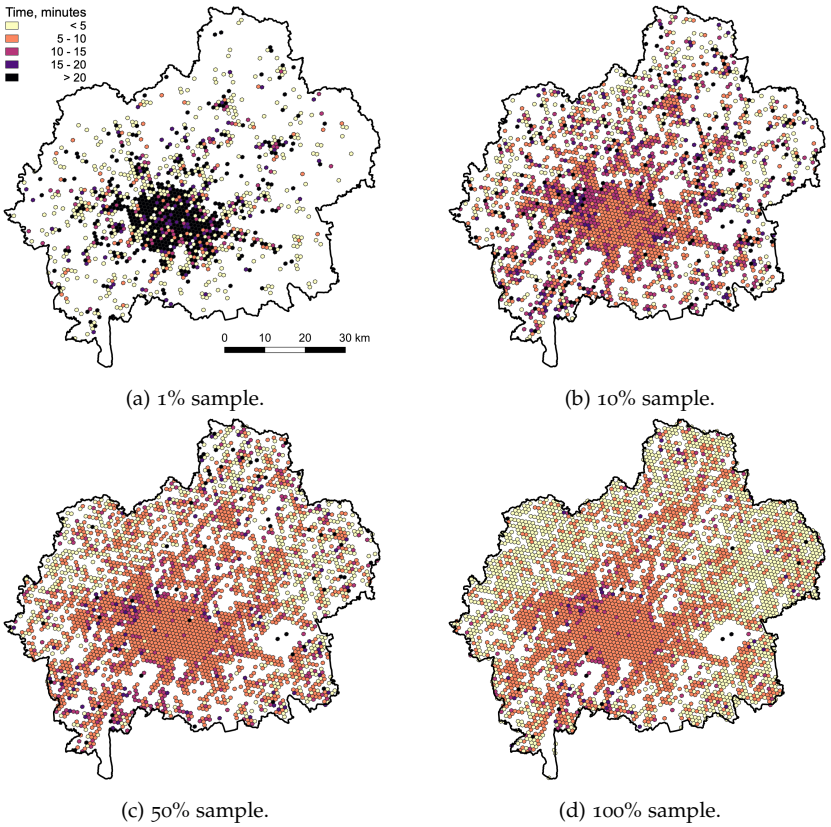


FIGURE 6.11: Impact of sample size on average per-vehicle daily driven distance in downscaled scenarios of the Munich area.

The spatial distribution of daily average waiting times is presented in Figure 6.12. Here, the locations of requests are aggregated in hexagons having an edge size of 500 m. It can be seen that the 1% sample size has a substantially different spatial distribution than for the full-scale case. One reason is that downscaling significantly reduces the number of requests in the low-density areas outside Munich. As a result, fleet vehicles are mostly concentrated in the city, where the vehicles generate and encounter more congestion, and thus for requests coming from low-density areas there is a lower likelihood of finding a nearby vehicle. Another reason for the difference is that traffic dynamics are changed more with smaller samples, causing different travel times. For the 10% sample size, the spatial distribution in the city is more akin to the full-scale case, while low-density areas outside of the city are still highly under-sampled.

The shift of the average waiting time during the evening peak hour, when the sample size is reduced from 100% to 90%, was examined in detail. In the current case, it seems that this shift arises due to a combination of multiple factors, including the spatial distributions of fleet and requests, the congestion patterns (in particular, the highly congested city centre and uncongested low-density surroundings), and the fleet size and served demand. The experiment was repeated three times using different sets of samples in order to exclude substantial stochastic variations in the simulation outcome; however, the outputs of simulations were observed to be similar for each set of samples. Further analysis of the case with the full-scale population showed that the equilibrium, which is finally reached, is relatively fragile, and even insignificant fluctuations in the non-taxi demand shifted the evening peak to earlier times, as in the case of the 90% sample. Therefore, even downscaling the scenario by 10% can lead to shifts of peak utilization of the fleet. That means that scenarios with fully utilized fleets must be very carefully evaluated during the downscaling process.

Overall, in comparison to the full-scale population, sample sizes of 30% or more predict fleet performance metrics that are of comparable, not exact, magnitudes.



(a) 1% sample.

(b) 10% sample.

(c) 50% sample.

(d) 100% sample.

FIGURE 6.12: Spatial distribution of daily average waiting times in downscaled scenarios of the Munich area.

These sample sizes of 30% or more can thus be used if only an approximate knowledge of spatially aggregated externalities of the fleet, such as noise or air pollution, are desired. However, as the predicted performance using downscaled inputs can have quite different characteristics during peak hours, such as a shift in the peak utilization, the use of the full-scale population is recommended if accurate spatial and temporal estimates are required. For example, with a sample size of 50%, the mean per-vehicle daily driven distance is 235 km; while with the full-scale population, the mean driven distance of 215 km is 8.5% less. Thus, for an all-electric fleet, with a German electricity price assumed to be €150/MWh and the average electric vehicle energy consumption of 0.2 kWh/km, the annual difference in cost for charging the fleet is about €3.3 million. Similarly, the required charging infrastructure and corresponding investments will be overestimated when

downscaled inputs are used in the mobility simulations. On the other hand, in low-density areas the charging infrastructure would be underestimated.

7

VISUAL ANALYTICS

What makes the desert beautiful is that somewhere it hides a well.
— Antoine de Saint-Exupery

While agent-based modelling is a promising approach to addressing existing and upcoming challenges in the transport systems of urban areas by using detailed and policy-relevant scenarios, some issues of ABMs have to be resolved before their wider adoption. One of such issue related to large-scale ABMs is data analysis and visualization.

First, agent-based scenarios generate large amounts of data that must be analysed. As simulation outputs come out in a disaggregated form where each agent can be tracked individually through the whole period of simulated time, an additional effort to aggregate and post-process the data is typically required. For that, faster storage solutions (software like appropriate databases [327], specialized file systems [328], or hardware solid-state drives) and parallel data processing [329] can bring significant improvements in runtimes. Second, the visualization and analysis of agent-based scenarios need to be adjusted to use individual data to understand traffic phenomena and their negative externalities better. Moreover, when using hardware accelerators like GPUs to run scenarios, visualization requires a special coupling of simulation outputs and scenario inputs as per-agent data is stored on a device in a way optimised for simulation execution rather than for analysis.

Although visualization comes as the last step in a whole simulation process, it remains one of the most important parts of the scenario evaluation after preparing input data and traffic simulation. Visualization provides a convenient way to display results for a user and allows them to explore data in multiple directions, that is, by executing queries to filter, aggregate or reduce a dataset, or interactively change the view of analysed outputs. Visualization becomes of special importance when highly multi-dimensional data is analysed, like agent-based traffic simulations where (i) data has temporal and spatial dimensions, and (ii) each agent has a set of individual attributes that affect their decisions. For example, by visualizing the movement of the agents on a map, one can identify that a certain public transit route becomes overcrowded because too many agents live and work in the same parts of a city.

However, when running a large-scale scenario, visualization may become a performance bottleneck if too many agents have to be displayed and analysed in spatial and temporal dimensions with a high resolution (that is, 1 second in time and 1 meter in space). Slow rendering of moving agents with unacceptable waiting times will result in a poor user experience with interrupting workflow and unwillingness to use such visualization software [330, 331]. As shown in Chapter 6, one approach

utilized by researchers is to scale down a scenario (that is, take 10% of the population and set flow capacities of roads to 10%) and multiply output results by a factor inversely proportional to a scale factor. While this will reduce the number of visualized agents, it may (i) also bias output results if insufficiently large samples are used, and (ii) retain a rather high number of agents in a downscaled scenario of extensive urban areas for smooth rendering of visualization frames.

Another area where big data analysis and visualization play a crucial role is the smart city concept [332, 333]. In a smart city, data is collected from distributed sensors, devices and citizens in order to find more optimal and efficient ways to operate the city and to improve people's daily lives. In large urban areas, this requires real-time big data analysis, visualization and monitoring of the city's assets and resources, and the same technologies used for agent-based simulations can be utilized in the infrastructures of smart cities.

Currently, there is a research gap in the area of the visualization of large-scale agent-based mobility simulations. First, most of the existing works are focused on static (no movement of agents) and aggregated forms of visualization like plots, charts or distributions. The lack of dynamics in the temporal dimension of a visualized simulation reduces users' ability for interactive reasoning about the causal effects of the situations on the roads. For example, dynamic visualization can help to quickly identify when and where traffic spillover situations occur without writing and executing complex data processing queries. Second, works focused on dynamic visualizations are typically applied for small-scale cases with up to a few thousand agents without quantitative scalability benchmarks. Third, almost no technical details have been provided on how to efficiently implement visualization of agent-based data, especially when simulations are running on external hardware accelerators with limited memory capacity and data transfer bandwidth.

This chapter aims to fill this gap, and proposes a visualization and analytics framework, Quartz, which is designed specifically to work with large datasets from agent-based mobility simulations.

7.1 BACKGROUND

One approach to visualizing large-scale highway traffic flows was introduced by Sewall et al. [117]. In the proposed method, each lane (discretized in cells) is represented as a continuum flow with a low variation in speed, and the discrete state of a vehicle is defined by the cell containing it. While this is not an agent-based simulation, it demonstrates how a traffic model can be defined to optimize visualization. In another study, Sewall et al. [118] presented a method to reconstruct and visualize massive traffic flows from discrete spatio-temporal data sources like sensors. This method searches for a trajectory for each vehicle in the whole state-time space, incrementally and depending on the vehicle's priority. The runtime performance was analysed with up to 500 vehicles, and it drops quickly with the growing number of discrete states (that is, the length of a highway). In a later

work, Sewall et al. [119] implemented a hybrid approach where a microscopic agent-based model is used in regions of interest and where the movements of agents are visualized, while the rest is modelled with continuum flows. On a road network of 2 152 km of lanes and with the demand of 110 000 to 190 000 vehicles, the agent-based approach provides approximately 18 to 8 frames per second, respectively.

Suzumura et al. [120] developed a large-scale traffic simulation platform that uses the parallel and distributed programming language X10 to accelerate computations. The platform was able to run a Japanese nationwide daily simulation with the demand of 180 000 trips per hour and with a network of 993 731 nodes and 2 552 160 links in 1.73 hours using 100 distributed computing nodes. While the authors mention that the presented platform includes a visualization part for moving vehicles, no performance metrics or implementation details related to the user interface have been provided.

Shen and Jin [121] presented an agent-based system for detailed traffic animation in an urban environment using the hardware-accelerated graphics engine Horde3D. While their work is focused on the realistic visualization of traffic flows without any analytical capabilities, it gives insights into the possible performance of such systems for large-scale scenarios. The traffic model includes a car-following model with different driving styles, continuous lane-changing models and detailed road infrastructure (multi-lane roads, traffic lights). A straight four-lane road with 40 000 vehicles takes about 40 milliseconds to calculate per frame, and 20 milliseconds per frame if the lane-changing model is disabled. The authors also notice that the performance is competitive when compared with another agent-based microscopic traffic simulator, SUMO [102], when using a similar scale.

Heywood et al. [90] used the FLAME GPU [91], an agent-based simulation framework, to run a microscopic traffic model on a GPU and visualize it in real-time. The visualization uses an OpenGL and Simple DirectMedia Layer framework to accelerate the rendering in a 3D environment, and CUDA OpenGL Interoperability is used to avoid unnecessary memory copying from the simulation part. This is one of few works where a GPU-accelerated traffic model has been coupled with visualization. The visualization was found to have a negative impact on simulation performance: specifically, for a small-scale artificial lattice road network with the grid size of 8 and 8 192 vehicles, it takes about 1.2 milliseconds per simulated iteration to visualize vehicles, representing an 8% decrease in the overall runtime.

Lu et al. [122] developed the Toolbox for Urban Mobility Simulations (TUMS) using LandScan [123] and OSM for input data and the microscopic agent-based traffic simulator TRANSIMS [101] to make it applicable globally (to any location). TUMS includes two levels of visualization: a macroscopic tool for link-based analysis (that is, congestion, delay and density) in 15-minute intervals, and a microscopic tool for vehicle-based analysis when individual vehicles are rendered using WebGL with up to 1-second resolution. Neither information about the rendering performance of the visualization tools nor implementation details were reported.

The MATSim framework [76] for agent-based transport simulations has multiple tools to visualize and analyse outputs from scenarios. A free, open-source tool,

OTFVis, has been developed mainly for the debugging of input data. OTFVis provides limited functionality for visualizing road networks, moving vehicles (recorded and in real-time) and highlighting results of executed queries in a real-time mode (that is, finding a link by ID or showing agents' routes). OTFVis uses OpenGL to accelerate rendering, but no aggregated analysis is supported. Another alternative is to use a commercial tool Via which provides many features for the visualization of individual agents and aggregated analysis. However, the free version is minimal in terms of the number of vehicles (up to 500) and functionality, and neither performance benchmarks for large-scale scenarios nor technical details of the tool are available.

Gehlot et al. [124] developed an agent-based and parallel evacuation simulator to handle complex decision-making processes in large-scale scenarios using microscopic traffic models. One of the work's main contributions is an efficient visualization module that communicates with the traffic simulator through WebSocket [334]. In each simulation tick, the simulator sends a list of moving vehicles to a collection queue to update on a screen, making it efficient to visualize as not all vehicles have to be updated. Per-link network statistics can also be collected during a traffic simulation and visualized in a separate viewer. The visualization module's computing performance was only evaluated qualitatively and is found to be smooth and efficient on a small-scale network of about 4 000 nodes and 8 000 links with up to 100 000 vehicles simulated. However, it is not clear if all vehicles were in motion at the same time.

Kim et al. [125] developed a mesoscopic traffic simulator with a real-time visualization library, SALT-Viz. The simulator can handle large-scale urban traffic simulations with dynamic generation of road geometry based on the attributes of links, and the visualization part uses OpenGL to accelerate rendering. The presented library uses overlays and multi-resolution rendering model to speed up the process. Additionally, 3D buildings can be reconstructed using rule-based modelling. The rendering performance was evaluated using the model of the Gangdong district of Seoul (South Korea) with a road network graph of 5 112 nodes and 12 437 links, and 370 093 vehicles. On average, the system was able to output 148 frames per second. However, the visualization part does not render moving vehicles but rather colourizes network segments based on flow density aggregated in fixed time intervals.

Charlton and Laudan [126] presented a web-based platform to visualize MATSim outputs. The entire user interface works in a web browser and can display aggregated (i.e., OD flows, per-link traffic volumes and emissions) and disaggregated outputs (movement of individual vehicles during a simulation). The performance of the presented platform is unclear. First, pre-processing time is stated to be from a few seconds to many minutes depending on the simulation scale. Second, WebGL's rendering performance of moving vehicles is stated to be enough to handle tens of thousands of vehicles, without explaining the size of the network or time required to render a frame. The platform currently follows the concept of good defaults where a user has minimum options to configure the "look-n-feel" of visualized datasets.

7.2 EVENTS

GEMSim is a discrete-time simulator with high spatial (1 meter) and temporal (1 second) resolution levels, and the traffic model is invoked in each simulated second. As a scenario has to be run for dozens and hundreds of iterations within the simulation loop, it is desirable to record required data for further analysis and visualization instead of doing it in real-time. For this purpose, discrete events were introduced into the simulation. These events are the same as in other discrete-event simulations [335] except that they are used only to record the simulation process, which means how the states of modelled objects have been evolving over time. For example, an event is emitted when an agent enters a new network link or when they start performing an activity after arriving at the desired location. This is similar to how coordinated fleets were simulated in Chapter 3, when events were used to notify fleet operators with riding requests from passenger agents, or state events from fleet drivers.

As described in Chapter 2, a different approach to storing data on a GPU is required for better performance throughput. For example, GPU-accelerated data structures can use indices instead of unique IDs to refer to agents in order to optimize runtime performance. Use of different data structures and the lack of object identification in the same way as in the input data (through unique IDs) on a GPU makes it impossible to simply collect and store data as it comes from mobility simulations. Instead, either a back-mapping process is required or additional metadata must be stored in order to make it possible to back-map recorded data later. The latter approach was chosen for analysis of events in GEMSim in order to avoid costly back-mapping process during a simulation. At the same time, some events, like those used by coordinated fleets, are back-mapped "on the fly" in a separate event stream, and the number of such events is orders of magnitude less than those recorded from a full simulation.

When events are recorded, the following rules apply:

- An event with a later time stamp always comes after an event with an earlier time stamp.
- Events of a single agent occurring in the same simulation time step are ordered according to the logical sequence. For example, in the first event an agent finishes an activity, and in the second event they board a vehicle.

A stream of events, organized in an ordered way, can easily be analysed and visualized as one can rely on the specific temporal sequence. A list of back-mapped events available in GEMSim is presented in Table 7.1. All events have a mandatory time stamp and a set of other fields that vary depending on the event type. Events also contain unique IDs of the related objects, and these IDs can be used during the analysis or visualization to filter, aggregate or build complex processing logic. In a typical simulation, `LinkEnter`, `LinkLeave` and `FuelChange` events correspond to

about 90% of all generated events as they are emitted each time an agent passes through a network link.

TABLE 7.1. Discrete events recorded by GEMSim during a traffic simulation.

Event	Description	Fields (except time)
LinkLeave	Vehicle leaves network link	Agent, link
LinkEnter	Vehicle enters network link	Agent, link
QueueStuck	Vehicle is stuck in traffic	Agent, link
ActivityStart	Agent starts activity	Agent, link, activity type
ActivityEnd	Agent ends activity	Agent, link, activity type
LegStart	Agent starts travel leg	Agent, link, transport mode
LegEnd	Agent ends travel leg	Agent, link, transport mode
VehicleBoard	Agent boards vehicle	Agent, vehicle
VehicleLeave	Agent leaves vehicle	Agent, vehicle
TransitArrive	Transit vehicle stops at station	Vehicle, station, departure, route
TransitDepart	Transit vehicle leaves station	Vehicle, station, departure, route
TaxiRequest	Taxi request comes from agent	Agent, start link, end link, operator
FuelChange	Fuel level of vehicle is changed	Vehicle, old level, new level

One of the limitations of GPU-accelerated simulations is that device memory must be allocated in advance. However, it is not known in advance how many, or when, events will occur during a simulation, and one cannot simply store all of the events on a GPU in a single chunk of memory to transfer them to the host afterwards. To overcome this limitation, GEMSim uses periodical data synchronization between a GPU and the host, in the same way as done for coordinated fleets. The memory on a device is pre-allocated in a separate buffer using the following assumptions:

$$M_{ebuf} = \frac{E_{max} \cdot S_{max} \cdot N_{agents}}{f_{sync}} \tag{7.1}$$

where E_{max} is the maximum number of events a single agent can emit in a simulation step (currently equals to 5), S_{max} is the size of the largest event data structure (currently equals to 24 bytes, and event structure is always aligned to the 8-byte boundary), N_{agents} is the total number of agents in the simulation, and f_{sync} is the data synchronization frequency between the host and the device. The number of agents N_{agents} also includes artificially created agents not provided in the input data (i.e., drivers of public transit vehicles). The memory buffer of size M_{ebuf} provides enough capacity to hold all possible events generated between two synchronization points. At a synchronization point, collected events are transferred from a GPU to the host and written into a file for further analysis. During a simulation, events are written into the buffer atomically, as described in Chapter 3.

A huge amount of data with events is generated during a large-scale simulation, and this stream of data must be handled almost in real-time so as not to slow

down the simulation process and to keep memory consumption at a reasonable level (as the event stream is coming from a GPU, it must be either written down or queued in memory for later processing). For example, when simulating the whole of Switzerland, about 22 GB of binary events data, or about 1 billion events, are generated in two minutes of real time; that is approximately 180 MB of data every second. This stream of data is compressed in a separate thread using the GZIP algorithm and written into a file, so the simulation does not wait at synchronization points for any time longer than is required to transfer the data. Here, another limitation of a GPU-accelerated model, mentioned above, comes into play: a GPU does not keep host-like data of the objects but rather indices to the objects in GPU memory are used by the device. For the same reason (high volume of data in real-time) the back-mapping process is not performed and the events are written as they come. To overcome this issue, a custom binary file format was used to include not only events, but also the corresponding metadata to perform back-mapping during the later analysis of the events.

A file with an event stream contains the following parts, written successively:

- **Header**. Information about the version of metadata and events that can be used for backward compatibility. The version of the metadata is increased each time it is extended with new information, and the version of events is increased each time new events are added into the simulation.

- **Metadata**. Back-mapping information required to convert events from a GPU representation to the host's and to link it with the scenario input data. These are key-value associative arrays where keys are GPU indices and values are host-based IDs or other data. For example, lookup arrays are built for the IDs of agents, network links, vehicles, public transit stops and fleet operators. There are also lookup arrays for non-ID values like transport modes, agent roles and types of activities.

- **Events**. A stream of time-ordered events as it comes from a GPU. The first field of any event is a 4-byte type which is used to properly parse it along a stream or to skip it, and each event has a static size depending on its type.

The header and metadata are written before the simulation begins, hence, even if a simulation aborts in the middle, the data which is already written can be parsed. The minimum amount of redundant data compared to other human-readable formats like XML allows it to save disk space, and a fast GZIP algorithm reduces the file size by up to a factor of three. There is, however, a separate utility, which comes with GEMSim, that can filter and convert binary events to the human-readable XML format for further post-processing in high-level languages like Python.

7.3 FRAMEWORK ARCHITECTURE

The overview of the Quartz architecture is shown in Figure 7.1. The main idea was to make the framework flexible, fast and scalable by using streamlined data

processing. The analytics pipeline comprises an event stream and a set of attached event handlers that are responsible for the analysis of specific areas of interest. For example, one handler analyses general travel statistics like per-agent daily travel time and distance, while another handler aggregates statistics related to coordinated fleet utilization. A stream parser reads events and concurrently back-maps them from a GPU-based representation to a CPU-based one using the metadata from the stream binary file. Each event from the stream is then fed into each handler where an event is either processed or discarded based on the handler's logic. In the end, each handler goes through the whole stream and filters out only relevant information. Input datasets are also connected to handlers such that an event stream can be analysed in a broader context (e.g., the physical length of road links is required to calculate statistics of travel distance). After the event stream is analysed, each handler outputs a hierarchical representation (a tree-like report item) of the results, and all report items are aggregated into a single structured report that is visualized for a user. Besides a report, input data from a scenario can also be visualized separately.

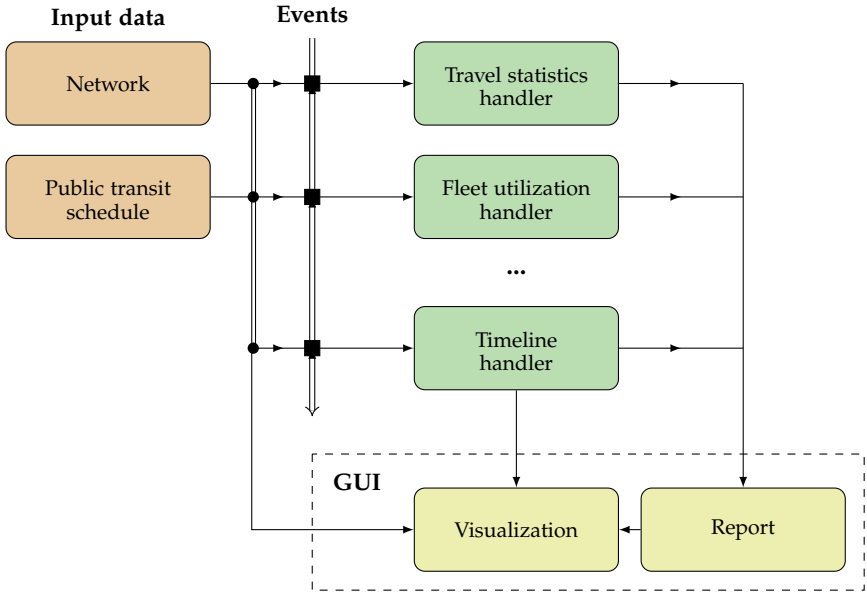


FIGURE 7.1: Architecture of the Quartz framework for data analytics and visualization of outputs from GEMSim.

To improve the runtime performance, a hybrid sampling approach was implemented. Hybrid sampling allows the model to use only a fraction of agents where it is acceptable, for example to build statistical distributions of travel time or distance. For visualization, it also makes sense to use only a sample of agents to keep

rendering smooth and not overload graphics with too many moving vehicles, as they are not rendered with a physical scale and overlap each other on a screen. The sampling rate for each of the event handlers can be specified by a user separately.

7.4 TIMELINE

The only handler that behaves slightly differently in terms of the output is the timeline handler. This handler builds a disaggregated timeline of agents' behaviour and outputs it directly to the visualization part of the framework without any report items. The purpose of the timeline is to describe how the state of each agent changes throughout a simulation, and visualize these changes (that is, movements or performed activities) individually. A timeline consists of frames, and each frame is generated at a simulated time step (1-second resolution) only if there were any changes of the agents. A frame includes the following information:

- **Trajectory points**. Each trajectory point specifies agent ID, network link ID and the time the agent spent moving along the link.
- **Activity points**. Each activity point specifies agent ID, network link ID, and the start time, duration and type of performed activity.
- **Arrived agents**. A list of agents who finished their trips and arrived at the locations of activities.

The timeline is one of the key parts of Quartz and it allows it to interactively visualize simulated traffic flows in motion. The reason why the timeline is split in frames with the transition of agents' states is because of how a graphics pipeline is organized in GPUs. In order to maximize GPU rendering performance and to visualize large-scale scenarios with millions of agents, the OpenGL framework is used to minimize computing effort on the CPU side and move as many computations to a GPU as possible.

The OpenGL framework works with GPUs through a graphics pipeline, and a simplified model of such a pipeline is shown in Figure 7.2. A host program prepares OpenGL input vertex buffers with the arrays of vertex attributes like position in space (x, y, and z coordinates) and colour or texture coordinates, and supplies the buffers into the vertex processing stage. At this stage, the GPU executes a vertex shader program for each set of vertex attributes and outputs an array of vertices projected into the screen space with modified (if needed) attributes. For example, a vertex shader can modify a colour attribute based on input variables like the simulation time. At the next stage, generated vertices are assembled into basic geometric primitives like triangles or lines, and are sent further to the rasterization stage. The rasterization clips and discards parts of the primitives outside the screen, and breaks remaining parts into pixel-sized fragments while interpolating vertex attributes across the rasterized surface of primitives. For example, a triangle is colourized with a gradient made from a colour value from each of the vertices.

Then, at the fragment processing stage, a fragment shader operates on each pixel and outputs individual values for colour and depth attributes. Lastly, after some final operations like depth testing and blending, the pipeline outputs a framebuffer that can be rendered on a screen. There can be some variations in the pipeline like additional shaders but the main principle and the workflow remain the same.

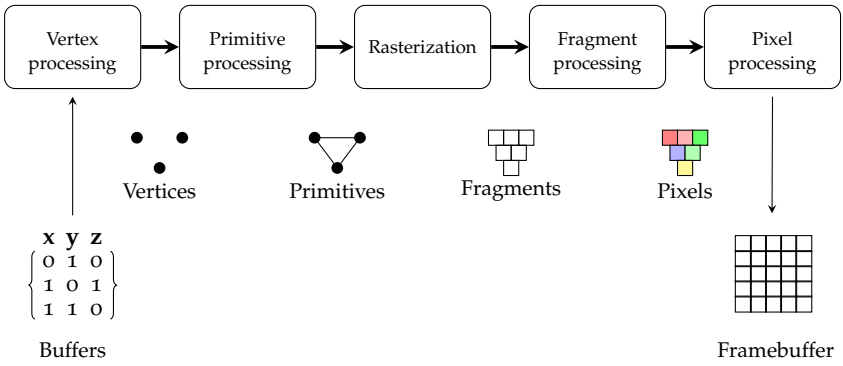


FIGURE 7.2: Structure of a typical graphics pipeline implemented by a GPU.

As modern GPUs have thousands of processing elements in a pipeline, they can execute shaders in a massively parallel manner achieving high computing throughput. Therefore, the main idea behind the timeline was to utilize vertex shaders to calculate and update the states of moving agents on the screen, offloading this work from a CPU. The timeline comprises two parts: (i) a controller part that is executed on the host side and (ii) a shader part which is executed on a GPU.

The controller part manages OpenGL vertex buffers and updates these buffers each time an agent changes their state (that is, finishes driving a link, or starts doing an activity). One of the issues here is that in each simulated second the number of agents in motion varies from a few to tens of thousands, and removing or adding agents for visualization can cause frequent re-allocation of OpenGL buffers. To avoid it, the timeline handler calculates the maximum number of agents that can be visualized in each second and pre-allocates buffers according to this number. Additionally, it maintains a list of free slots in the buffers according to the following rules:

- When an agent arrives at the location of activity, the corresponding vehicle is marked with a transparent colour, and the slot (index) in the OpenGL buffers is marked as free.

- When an agent starts travelling, the first available slot in the buffers is marked as used, and the personal attributes of the agent are used to fill data in the buffers.

This approach allows the size of the buffers to be kept constant, and a GPU performs the work to discard invisible objects from the screen. As this procedure is implemented in hardware and executed in parallel, it provides a much higher runtime performance than if it were executed on a CPU.

The timeline supports variability in visualization speed, which means how many seconds of simulated time to render in one second of real time. The update frequency of the buffers depends on the visualization speed-up compared to real time; however, as the temporal resolution of a simulation is no less than a second, no update of OpenGL buffers is required in between timeline frames on the host side. For example, if the speed-up factor of visualization is set to two, then, in each second of real time, the timeline may receive up to two frames (each within a half-second) with agents to update, and will redraw the screen an additional 46 times to make the movement of the vehicle smooth with 24 frames rendered per second.

Between any two frames, the shader part of the timeline is executed on a GPU to animate moving vehicles. For each moving vehicle, the vertex shader receives, in arrays, the start and end locations of a network link, the start and end time stamps for the movement along the link, and a colour for the vehicle. Before executing the shader program, the timeline controller updates the current simulated time, and the shader can then interpolate the position and the direction of a moving vehicle (marked as a triangle) along a link. Here, data in vertex buffers is represented by trajectory points from the frames, and CPU does not need to perform additional calculations.

7.5 USER INTERFACE

The Quartz framework was developed as a native application in C++ using the Qt cross-platform framework and can run on Windows, GNU/Linux and macOS. The main window of the Quartz graphical user interface (GUI) with the Hokkaido (Japan) scenario visualized is shown in Figure 7.3. Here, each vehicle is represented as a directed triangle, where green marks private cars, red is for public transit vehicles, and purple dots indicate the locations of activities. The speed of visualization as well as the simulated time can be controlled interactively at the bottom bar.

The window is split into three areas: a map view on the right, a list of geo-referenced layers to render on the left, and tabs to switch between the map and reports on top. Layers are rendered in the map view according to their vertical z-order, one on top of another, but any layer can be disabled from rendering. Plots, as well as the map view, can be interactively zoomed and panned while the dynamic visualization is running.

The reporting part of the interface is presented in Figure 7.4 For now, a report can contain the following analytics:

- **Travel statistics**. Aggregated statistics like distributions of travel distance and time, and average and total values for VKT and VHT (Figure 7.4).

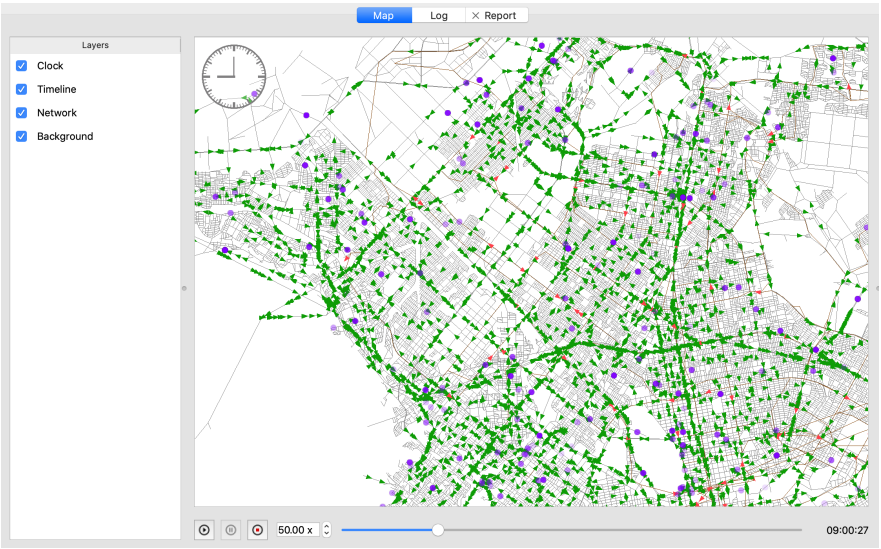


FIGURE 7.3: Main window of Quartz for the Sapporo area from the Hokkaido scenario.

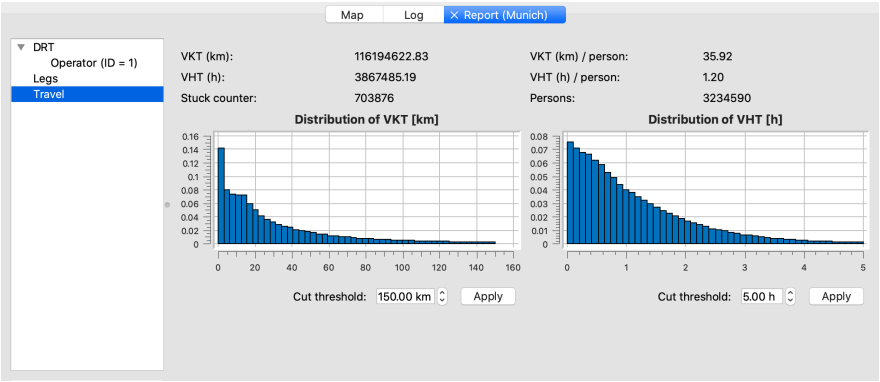


FIGURE 7.4: Overall reporting interface of Quartz with the travel statistics section for the Munich scenario.

- **Mode statistics**. Aggregated statistics for each mode of transport, including departing, arriving and en-route agents, and the total number of travelled legs and legs per person (Figure 7.5).

- **Fleet utilization**. Provides aggregated information about coordinated fleets of vehicles, including empty mileage, idle time, occupied time and the dis-

tribution of driven distances (Figure 7.6). Each fleet operator is processed separately in case a multi-fleet scenario was run.

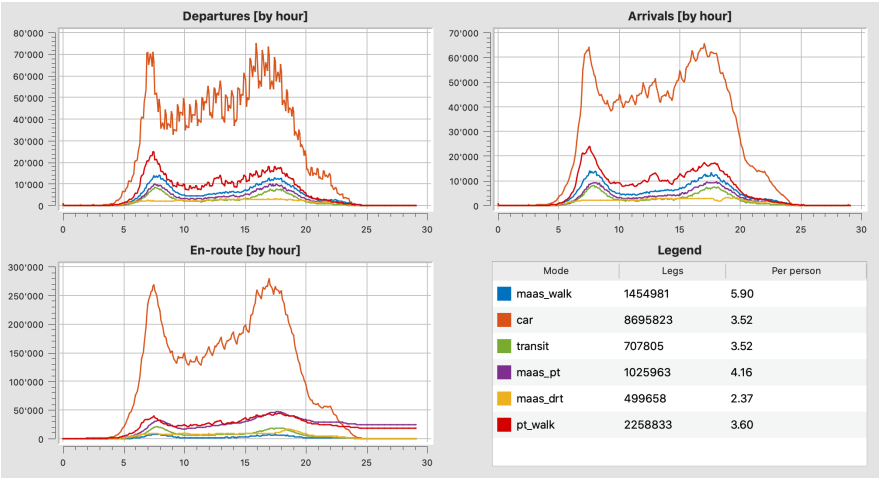


FIGURE 7.5: Reporting interface of Quartz with travel mode statistics from the Munich scenario.

Some of the plots provide interaction with data, for example a user can select a cut value for histograms. Each layer can provide options to configure its behaviour and visualization, for example changing colours or the way the colours are assigned. Figure 7.7 shows a customized visualization of the Los Angeles area (USA).

Another important capability of Quartz is data filtering, when original events can be narrowed down to a subset based on provided criteria. Filtering allows running further analysis and visualization only to the area of interest, hence improving runtime. Figure 7.8 shows Quartz filter options, of which any combination can be selected. The event filter was implemented as another event handler in the processing loop, and it analyses events in two passes. In the first pass, it collects information about which agents do fall into filtering conditions, and in the second pass it actually discards unrelated events and write the output stream. This also demonstrates the flexibility of the streaming approach implemented in Quartz.

7.6 BENCHMARKS

To benchmark the Quartz framework, two large-scale scenarios were used: the Switzerland scenario from Chapter 4 and a scenario made for the prefecture of Hokkaido in Japan based on available travel statistics [336, 337]; a summary of the scenarios is presented in Table 7.2. Due to the limited availability of disaggregated travel data from Japan [338], only aggregated statistics were used to generate the

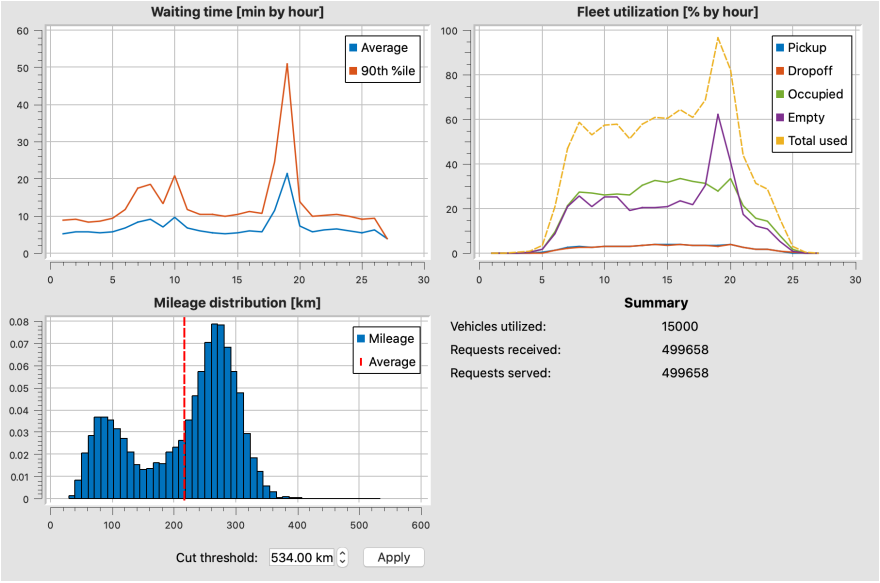


FIGURE 7.6: Reporting interface of Quartz with fleet utilization from the Munich scenario.

FIGURE 7.7: Interface of Quartz with a customized visualization of moving agents in the Los Angeles area from the California (USA) scenario.

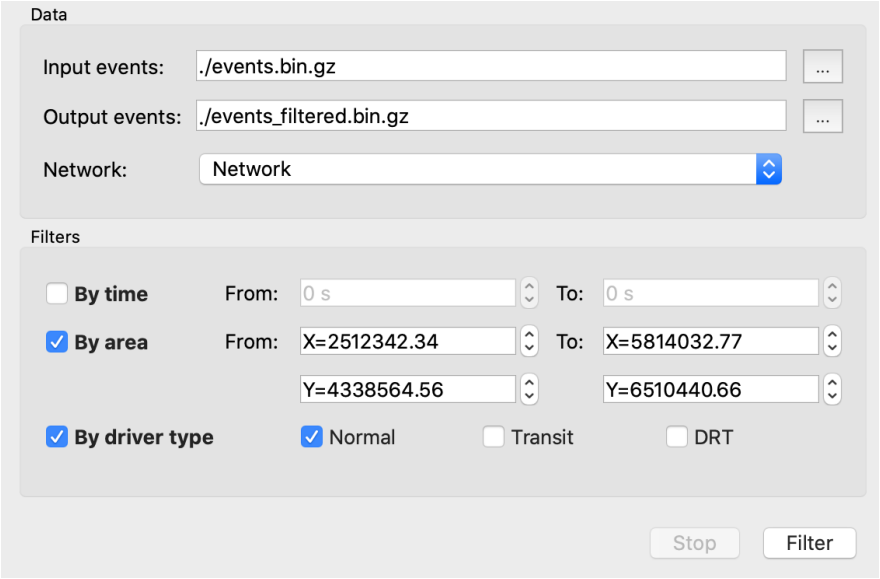


FIGURE 7.8: Interface of Quartz with filtering options for a stream of events.

Hokkaido scenario. Both scenarios have comparable road network sizes, however the Switzerland scenario has a full national public transit schedule incorporated while the Hokkaido scenario has a transit schedule limited to the area of Sapporo. The Switzerland scenario has almost double the number of agents and events (more than 1 billion events are generated), hence the number of events almost linearly scales with the number of agents, having the same road network size. Scenarios were run for 200 iterations to converge and the output events from the last iteration were used for performance benchmarks.

TABLE 7.2. Summary of large-scale scenarios used for Quartz performance benchmarks.

Scenario	Agents	Events	Data size, GB
Switzerland	5 542 305	1 344 502 163	21.09
Hokkaido	2 924 938	739 452 279	13.06

A laptop with an Intel Core i9-9980HK CPU, AMD Radeon Pro 5500M GPU, 32 GB of RAM and an SSD was used to run Quartz with simulation output events. First, the runtime performance of data analytics is evaluated by using output events from both scenarios; handlers for general travel statistics, mode statistics and timeline were enabled. Each scenario was run with a sampling rate for travel statistics and timeline from 10% to 100% with a step of 10% to assess the scalability

of the framework. The sampling was performed by Quartz, while the scenarios were always run at the full scale. Figure 7.9 shows the runtime required by Quartz for data analytics, and the runtime scales sub-linearly depending on the sampling rate and the size of a scenario. While the overall population in the Switzerland scenario is almost twice as large as the Hokkaido scenario, the overall processing time is more than double. For the smaller scenario of Hokkaido, the runtime also increases more slowly with the increase of sampling rate. This sub-linear scalability reflects a non-linear logic implemented in event handlers, when unrelated events are being discarded efficiently.

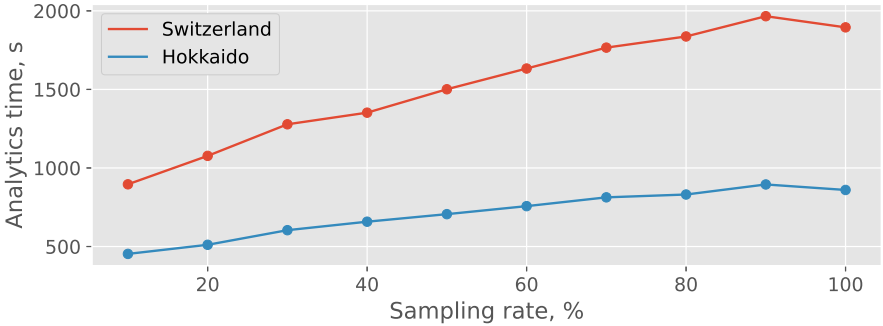


FIGURE 7.9: Runtime performance of data analytics by Quartz depending on sampling rate.

The consumption of host RAM by Quartz when performing data analytics is shown in Figure 7.10. Here, the RAM consumption grows more linearly with the number of agents to be analysed. Most of the RAM is consumed by the timeline to store the frames, while the rest of the handlers store only limited amount of data, some of which is already aggregated. Nevertheless, 10% sampling rate consumes less than 2 GB of memory which can be handled easily on commodity hardware.

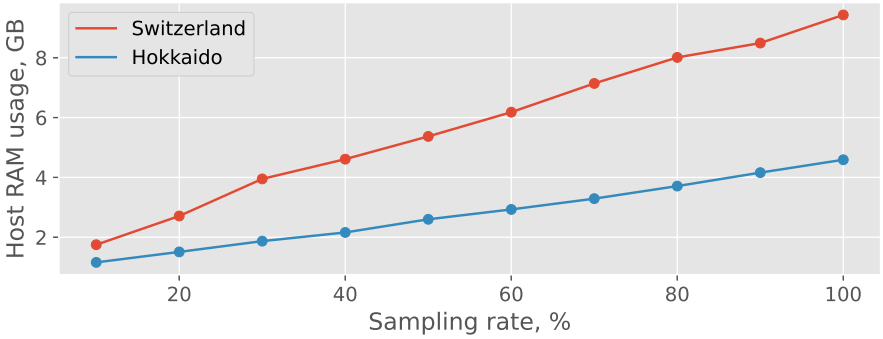


FIGURE 7.10: Peak host RAM consumption for data analytics by Quartz depending on sampling rate.

Finally, the runtime performance of the timeline was evaluated. Figure 7.11 shows the runtime required to process a single frame depending on the number of agents that have their states updated in this frame. As one can see, 2 000 agents can be processed at a rate of about 20 frames per second, which is enough for smooth rendering. When the number of agents in a frame increases to 3 000–4 000, the frame rate drops to about 6–10 per second. However, one must consider that for a typical scenario, the number of agents per frame varies a lot during the simulation, hence only in a few peak moments might it have 4 000 agents, while for the rest of simulation it will be 2 000–3 000. One should not confuse the number of agents updated in the frame with the number of agents being rendered; in the latter case the number of agents can be orders of magnitude higher.

The experiments show that 4 000 agents at the peak correspond to the 50% sample of the Hokkaido scenario, which means that about 120 000 of the agents were visualized simultaneously during the peak hour. Increasing the number of agents in a population sample leads to visible slowdown of rendered movement on the screen due to long frame processing times. Therefore, scenarios with up to 2 million agents can be visualized directly without filtering, while for larger scale scenarios pre-filtering is required for dynamic visualization.

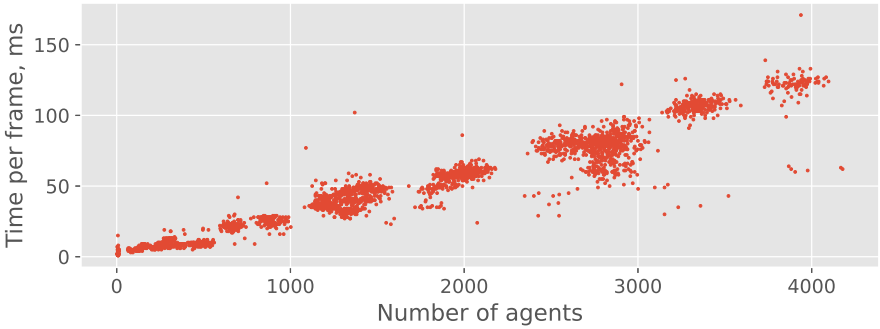


FIGURE 7.11: Time required to render a frame of the dynamic visualization timeline in Quartz.

It should be also noted that, while the timeline performs rendering, it remains fully interactive, and a user can pan and zoom on the view, adjust rendering speed and customize the visualization properties of the layers. This should improve user experience while working with large-scale datasets from simulated scenarios. As the developed analytics and visualization framework has flexible architecture, other analytical modules and geo-referenced layers can be integrated into Quartz easily.

8

CONCLUSIONS AND OUTLOOK

Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.
— Antoine de Saint-Exupery

8.1 CONCLUSIONS

8.1.1 *Modelling*

GPU-accelerated mobility modelling framework. In this thesis, an agent-based GPU-enhanced mobility simulator (GEMSim) framework was developed. The framework incorporates a co-evolutionary learning process in its agents, when they adapt their behaviour between simulated daily iterations based on previous travel experience. A simulation converges when unilateral improvements in the travel behaviour of the agents are negligible. The framework provides a solid and ready-to-use foundation for the modelling of large-scale scenarios with agents' complex behavioural logic and detailed transport infrastructure. The developed traffic model is based on the mesoscopic spatial queueing model that provides a trade-off between the modelled level of detail and runtime performance. GEMSim was developed in a modular way to allow the further extensions and adaptations required to simulate future mobility scenarios. The massively parallel, GPU-accelerated traffic model provides substantial speed-up in simulations – being up to two orders of magnitude faster – compared to MATSim, a state-of-the-art agent-based transport modelling framework with a conceptually similar traffic queueing model. To simulate the traffic propagation of a typical day for the whole of Switzerland, using the full population sample and the full-scale road network, takes GEMSim less than 2 minutes. The whole simulation loop of GEMSim, with the learning process included, runs more than 22 times for the same scenario of Switzerland, compared to MATSim. These runtime performance achievements prove that GPUs are well suited for general-purpose agent-based traffic simulations with the complex patterns of behaviour used by the agents.

Gridlock resolution. In order to make GEMSim's GPU-accelerated traffic model scalable and applicable to a wide range of mobility scenarios, a gridlock resolution is required. Without a gridlock resolution strategy, a simulated scenario may produce unrealistic congestion levels with completely clogged areas. While there are multiple methods used in other CPU-based traffic models, these methods rely on the fact that the host memory can be dynamically allocated during the simulation process, which is not the case for GPUs. To tackle this issue, novel gridlock resolution strategies,

which can be also applied to CPU-based traffic queuing models to improve runtime performance, were proposed and implemented. These strategies allow it to efficiently, in terms of the impacts on runtime and the fidelity of simulated outputs, resolve gridlock situations that tend to happen when queueing traffic models are used, especially at a large scale.

Multi-modal extensions. Multi-modal extensions to simulate public transit and coordinated fleets on GPUs, integrated together with car traffic, were implemented in GEMSim, making it capable of running large-scale and multi-modal scenarios. It was found that intensive interactions of passenger agents with vehicles at the stop facilities when waiting for public transit or taxis, together with hardware limitations of GPUs and CPUs, require a different modelling approach to mitigate an otherwise significant runtime performance drop. A novel state-based GPU-accelerated approach was developed to overcome these limitations. Moreover, simulation of fleets required redistribution of computing workload among GPU and CPU sides to achieve optimal runtime performance. For public transit, a speed-up factor of 31 was achieved compared to MATSim, while for a coordinated fleet of 100 000 vehicles a speed-up factor of up to 50 was achieved.

8.1.2 *Hardware*

Heterogeneous CPU-GPU hardware. As modern computing, especially high-performance systems, is becoming more heterogeneous, with various hardware accelerators attached including GPUs, a novel approach to running mobility simulations on heterogeneous hardware with GEMSim was implemented. The whole simulation loop was designed to fit and to optimally distribute heterogeneous computations, between both the host and the device sides of the hardware. First, a hardware abstraction layer separates the whole simulation loop from the parallel execution of the traffic model on specific hardware, so that only a relatively small part of the simulator, called the backend, interacts with hardware directly. Second, a data binder abstracts data layer management from specific hardware by applying a device-optimized memory layout for host data structures. Both backend and data binder concepts allow to port GEMSim onto other types of parallel hardware with little effort. Such portability was demonstrated by running the same large-scale mobility scenarios on many-core CPUs, including ARM architecture, without any GPU acceleration. While the CPU backend is capable of running the traffic model up to 4 times slower compared to a GPU backend, it can still deliver a speed-up of more than 11 times compared to MATSim running the same scenario with the same hardware resources. An interesting finding was that the CPU backend could bring more commercial benefits when running large-scale scenarios in the cloud. Here, GEMSim provides a trade-off between the cost and runtime of simulations: while the CPU backend is 4 times slower than the GPU backend, the price of GPU-accelerated instances in computing clouds can be 6 or more times higher. In the long term, however, the GPU backend can bring larger energy savings than the

CPU backend, consuming about 2.76 times less energy to run a large-scale scenario. It is also interesting that older CPU-GPU hardware is even more energy-efficient. Overall, a mobility model which supports heterogeneous hardware can provide more flexibility in terms of the cost-benefit ratio. This flexibility allows a user of such models to adapt easier to the dynamic availability of hardware on the market and gives more options for long-term investment decisions related to hardware.

Runtime performance bottlenecks. In general, large-scale agent-based mobility simulations have two major bottlenecks in runtime performance: traffic propagation through spatial queues and the learning process. The developed GPU-accelerated model resolves the former bottleneck by providing up to 100 times faster traffic propagation model. However, custom hardware accelerators generally do not have coherent memory space shared with CPUs and require data to be transferred back and forth between an accelerator and a CPU. Data transfers form another, the third, runtime performance bottleneck, although of a much smaller magnitude than the traffic propagation part when running on CPUs. For example, when running a 10% sample of car traffic in Switzerland, data transfers and traffic propagation contribute about 8% and 61% of the runtime, respectively. Still, for the full population sample, the same parts of the simulation loop contribute about 17% and 43% of the runtime, respectively. An interesting finding of the work is that in a GPU-accelerated model, the learning part becomes the largest runtime performance bottleneck contributing about 40% of iteration runtime at a large scale. Currently, GPUs are not well suited for routing algorithms used during the learning process, but the situation may change with the switch to more complex behaviour modelling like reinforcement learning (RL).

8.1.3 *Applications*

Travel demand generation. As travel demand is one of the key inputs for any agent-based mobility simulation, this thesis had a goal to improve demand generation for large-scale scenarios. A unified modelling pipeline was implemented and validated with the case of Switzerland. The pipeline represents a sequence of data transformation steps, including car ownership and transport mode discrete choice models, which propagate input data until they produce individual daily-activity plans for the agents. It was demonstrated that GEMSim can accurately reproduce the Swiss cars' and public transit's travel time and distance distributions found in the national mobility microcensus using the developed pipeline. Moreover, as the pipeline is highly parallelized with multi-core CPUs, only about 2 hours are required to generate travel demand. Furthermore, the pipeline has an integrated mechanism to flexibly adjust the behaviour of the agents, as was shown for the case of COVID-19 measures in Switzerland. It should also be noted that the same approach with the unified pipeline has already been applied to other countries and regions, such as Bavaria (Germany) and California (USA). However, the quality of the generated agent-based travel demand heavily depends on the quality of

available datasets, and disaggregated data from travel surveys is one of the keys for agent-based mobility scenarios.

Coordinated taxi fleets. Multiple case studies were conducted using GEMSim and developed demand generation pipelines for Switzerland and Germany. The first case study evaluated the impacts of the deployment of a coordinated fleet of automated vehicles (AVs) in the Zurich area of Switzerland to substitute the travel demand of car users with private taxi trips. The results showed that a replacement rate of seven to eight can be achieved if passengers are willing to wait up to 10–15 minutes during the evening peak hour. However, the fleet deployment will increase the total vehicle-kilometres travelled (VKT) up to 25% in the area, leading to vehicle-hours travelled (VHT) increased by 44% on average, from 39 to 57 minutes. Overall, an AV fleet is likely not to be a good candidate for the wide replacement of private cars in the city of Zurich, at least until the driving efficiency of AVs is improved. The second case study evaluated the potential acceptance of Mobility-as-a-Service (MaaS) in the Munich metropolitan region of Germany. The proposed MaaS comprises a tightly coupled public transit system as a backbone mode and a fleet of private taxis to serve the first and last mile travel legs of the trips. This study not only demonstrated GEMSim's multi-modal capabilities for emerging modes, but also assessed the impacts of geo-fencing limitations (put on legs travelled by taxi) on the performance of MaaS operators and customers. It was found that more flexible operating policies for fleets, like longer first and last mile travel legs, can bring more benefits for operators and customers. However, a lack of restrictions on fleet operators can lead to the increase of negative externalities like traffic congestion (empty mileage is about 20%–40%) due to the shift of demand towards the agents who benefit from longer taxi travel legs and more frequent service usage during the day. In general, a fleet of 10 000 vehicles with taxi travel legs of up to 3 km can attract 7%–15% of the travellers with an average waiting time of about 8 minutes and up to 18 minutes in a peak, while utilizing the fleet for about 75% during the day. The majority of MaaS adopters are existing public transit users, and without additional incentives car users are unlikely to switch to MaaS or public transit. It was also found that about 80%–85% of MaaS users are living in the areas with high accessibility to public transit, so if policymakers want mobility concepts like MaaS fleets to succeed in reducing car usage, investments in public transit infrastructure are required.

Downscaled agent-based mobility scenarios. The developed mobility simulation framework was used to study the impacts of downscaled scenarios on simulation outputs. A novel measure of the goodness-of-fit of output results from two scenarios was proposed and evaluated together with other, more commonly used measures. The proposed measure is based on well-known chi-squared statistical distribution. The queueing model, used in GEMSim, was found to be more sensitive to the scaling of capacity buffers rather than to spatial buffers; this knowledge, however, can be used for calibration purposes to reduce the search space of the optimal buffer scaling coefficients. It was found that there is a critical size of approximately 30% of the full population, below which the simulated traffic dynamics are markedly different, as

well as the dynamics of the occupancy of public transit vehicles. However, if high accuracy in the predicted traffic dynamics and in the occupancy of public transit are not required, and the main interest is in only aggregated parameters such as VKT and/or VHT, with errors in the range of 2%–6% being acceptable, then smaller population samples, in the range of 5%–10% of the full population, can be used. For coordinated taxi fleets, sample sizes of 30% or more predict fleet performance metrics that are of comparable, though not exact, magnitudes. These sample sizes of 30% or more can thus be used if only an approximate knowledge of spatially aggregated externalities of the fleet, such as noise or air pollution, are desired. However, as the predicted performance using downscaled inputs can have quite different characteristics during peak hours, such as a shift in the peak utilization, the use of the full-scale population is recommended if accurate spatial and temporal estimates are required.

Large-scale visual analytics. A complementary framework for visual analytics of agent-based simulations, Quartz, was developed during this thesis. Quartz provides a flexible and extensible architecture that can analyse and visualize a stream of discrete events recorded by GEMSim during a simulation. The tool provides customizable and interactive visualization of the movement of the agents, as well as generation of analytical reports with various metrics like travel time and distance distributions, or fleet utilization breakdown. GPU acceleration is used to dynamically render agents in real-time, without significant drop of frame rate, from scenarios with up to 1.5–2.0 million agents. For larger scenarios, Quartz provides sampling and filtering capabilities that can reduce the number of agents to be processed. Working with discrete events coming from GPU-accelerated agent-based models, however, requires additional data conversions to keep short runtimes of the simulations. The main reason is that input data is stored differently in GPU memory, and one has to save additional metadata allowing later required data conversions for visual analytics.

8.2 OUTLOOK

8.2.1 *Modelling*

Non-motorized modes of transport. In addition to cars, public transit and coordinated fleets, other modes of transport like bicycles or scooters can be implemented given that many cities promote sustainable travelling. For these modes, the input multi-modal network will require some adaptation to include dedicated lanes and pedestrian areas, as well as modification of the queueing mechanism. The problem is that bicycles and scooters can seep between cars and avoid congested areas, therefore either the queueing approach needs to be modified to account for the flow capacity of non-motorized modes separately, or such behaviour should be modelled through separate queues. For battery electric variants of bicycles and scooters the power consumption model can also be adapted. Urban air mobility modes using electric

vehicles with vertical take-off and landing can be also implemented, benefiting from the available computing power of GPUs for in-air navigation algorithms.

Shockwave back-propagation model. To improve the fidelity of modelled traffic dynamics, a shockwave back-propagation model can be implemented. In the current implementation, as soon as a vehicle leaves a link's spatial buffer, the previously occupied space, located at the front of the queue, becomes immediately available to vehicles entering the same buffer from the upstream links. In reality, however, newly available space back-propagates through the queue of vehicles, affecting the congestion formation and spillover effects. On GPUs, this back-propagation behaviour can be implemented using double queues for each of the network links: one queue for the forward propagation of traffic, and the other for the backward propagation of free space. Such implementation will require double the memory used for networks on GPUs (which is a few hundred megabytes only for large-scale scenarios). It should not significantly increase the runtime, as the performance analysis showed that the GPU spends most of the time in the demand scheduling kernel, and not in the kernel that processes link buffers.

Hybrid traffic model. In the thesis it was demonstrated that the traffic propagation model leaves the computing units of a GPU heavily underutilized. Consequently, it is possible to perform more computations while waiting for memory transactions to be completed. For example, it is possible to implement a hybrid traffic model, when some areas of interest are modelled with a higher level of detail using a microscopic traffic model. This can be especially useful for urban and rural areas, where rural areas around a city define boundary conditions through incoming and leaving traffic flows, while the urban part is modelled with more detail.

Traffic lights. Currently, the impacts of traffic lights on traffic formation and congestion are not fully accounted for. This is partially compensated for through the reduced flow capacities of the roads, and partially through the calibration coefficients of link buffers. The GPU-based implementation of traffic lights can be relatively straightforward, when scheduled light cycles are stored either directly on a GPU or, as with network variations, at the host side. Then, at each simulation time step, another kernel in the GPU-based simulation loop can check traffic light cycles and flag network links to block the movement of vehicles. This can be done in parallel when each GPU thread processes a traffic light location. In case one also wants to model traffic lights' dynamic schedules, these checks should be moved to the host side as more complex logic could be involved, such as RL.

Parking space. Implemented parking capabilities are quite limited, involving sending agents to some network links to perform a specialized parking activity. This, however, does not facilitate proper occupancy tracking, or the application of various payment schemes depending on who wants to park and where. Parking space is of special importance for the potential deployment of coordinated fleets with AVs, because there should be a trade-off on how much public space can be reclaimed and how much still is required for fleets to keep empty mileage at an acceptable level. A parking model can be mostly host-based, as on GPUs an agent does not need to know the real reason they are going to a certain location.

Reinforcement learning. One can identify the areas of mobility simulations to which RL is applicable, for example, agents' decision-making, or selection of the locations of activities. As GEMSim already runs on GPUs, the same GPUs can be used to execute RL algorithms in an efficient way. While such a simulation loop may require many more iterations to converge, it can provide a more sophisticated decision-making process. A similar approach can be also applied to infrastructure like the dynamic optimization of traffic lights, or pricing for recharging infrastructure. One can also think about using multiple GPUs so that one runs traffic simulation while another is used for RL algorithms.

8.2.2 *Hardware*

Multi-GPU traffic model. One of the current limitations of GPU-based models, in general, is a relatively small amount of available on-board memory, which is typically up to 80 GB per single board for the top GPU models on the market. Estimates show that more memory is required to run large countries like Germany or France in their entirety, without breaking a scenario into sub-regions. The architecture of GEMSim allows the implementation of such an approach, but it may require substantial changes in the simulation loop. First, one should implement partitioning of the population into realms, each of them running on a separate GPU. Second, one has to deal with the limitations in dynamic memory allocation on GPUs. As agents can cross the boundaries of realms during the simulation, their data, like individual daily plans, need to be transferred between GPUs. This transfer procedure can be even more complicated for public transit vehicles or shared taxis, as passenger data must be collected and transferred as well. Potentially, an approach similar to dynamic agents for taxi drivers can be utilized for transferred agents. It is, however, unclear what the impact would be on the runtime performance when running such a multi-GPU traffic simulation. One can also consider using Nvidia's Unified Memory technology, which allows GPUs to transparently access each other's memories while the devices are connected directly through the high-speed NVLink interconnect.

Unified Memory. As the CPU and GPU parts of the simulator require tighter synchronization when modelling fleet services, one may think about using a so-called Unified Memory provided by CUDA. This technology allows the use of a single memory space for all devices in a system (CPUs and GPUs), and automatically migrates data between the host and a device when required (i.e., when a GPU thread attempts to access an array which is located on the host side). The Unified Memory can greatly simplify the development of such GPU-accelerated models by taking over some data copying responsibilities from a developer, as well as giving more opportunities like multi-GPU execution without implementing complex cross-device synchronization schemes. It would be interesting to evaluate the applicability of this technology to traffic simulations, as well as the impacts of introduced data management overheads on the runtime performance. The Unified Memory design can also include cases when CPUs and GPUs share the same physical memory space,

so the data is always coherent. Execution on such coherent architectures would be of especial interest, and one may also consider splitting the traffic modelling workload between CPU and GPU threads. Moreover, emerging technologies like CXL (Compute Express Link) may provide memory pools for both CPUs and GPUs, hence solving the issue of limited on-board memory capacity.

Data types with reduced precision. As was demonstrated, GEMSim's GPU-accelerated model is mostly memory- or latency-bound in terms of runtime performance. Therefore, in order to improve the performance even further, one needs to either improve the access patterns of GPU threads (which was shown to be difficult for agents with non-linear logic), or reduce the amount of data transferred to and from global (on-board) memory. Recently, GPUs gained support for data types of reduced precision used mainly in machine learning applications. For example, instead of using a four-byte data type to store a floating-point variable, a two-byte data type may be suitable in some cases, effectively reducing the required memory bandwidth by half. This approach, however, may face difficulties and runtime penalties due to the need for data type conversion between the host and the GPU.

More non-CUDA backends. With the ongoing developments in heterogeneous computing, more hardware engineering companies are releasing frameworks for software developers to simplify the implementation of massively parallel models on heterogeneous hardware. For example, Intel is pushing its oneAPI toolkit which allows a unified code to run on CPUs, GPUs and other types of hardware accelerators. At the same time, AMD also provides the HIP programming environment which allows portable GPU-accelerated applications that can run on GPUs from both Nvidia and AMD. Another company, Apple, has its own application programming interface, Metal, used for hardware-accelerated graphics and massively parallel computations. Thus, it would be interesting to see GEMSim (and other GPU-accelerated mobility simulators) ported to other hardware and investigate its runtime performance there.

Vulkan rendering in Quartz. The OpenGL interface, used by Quartz to render moving agents, is already 30 years old and it was not designed for modern hardware. One of the most serious limitations of OpenGL is single-threaded rendering, where only a single CPU core can interact with the rendering context. Modern rendering interfaces like Vulkan or Metal were designed for multi-threaded systems, and they provide possibilities for low-level optimizations like multi-threaded rendering. One can try to use such new interfaces to improve the rendering performance of Quartz when multiple threads can prepare per-frame data.

A

APPENDIX (DOWNSCALED SCENARIOS)

A.1 CALIBRATION COEFFICIENTS

TABLE A.1. Optimal scaling coefficients for spatial and capacity buffers of the network obtained for different population samples of the Switzerland scenario.

Sample, %	$\overline{k_l^*}$	$\overline{k_f^*}$	$\sigma[k_l^*]$	$\sigma[k_f^*]$
1	0.0086	0.0110	0.0025	0.0012
2	0.0680	0.0200	0.0192	0.0000
5	0.0920	0.0500	0.0130	0.0000
10	0.6200	0.1000	0.2588	0.0000
20	0.9000	0.2200	0.1414	0.0447
30	0.9400	0.3000	0.0548	0.0000
40	0.9600	0.4000	0.0548	0.0000
50	0.9800	0.5000	0.0447	0.0000
60	0.7800	0.6000	0.2049	0.0000
70	0.5600	0.7000	0.0548	0.0000
80	0.7000	0.8000	0.0707	0.0000
90	0.8000	0.9000	0.0707	0.0000

A.2 MEASURES OF GOODNESS-OF-FIT

TABLE A.2. Measures of goodness-of-fit for the morning peak hour (07:00–08:00) obtained for different population samples of the Switzerland scenario. Values of the measures of the smallest population samples passing an acceptance threshold are marked in bold.

Sample, %	U	U_s	U_m	U_c	$RMSNE$	GEH_5, %	GEH_{10}, %	S_α^m	$S_\alpha^{m,hw}$
1	0.1970	0.0022	0.0113	0.9864	2.7663	19.77	13.58	0.51	0.23
2	0.1441	0.0021	0.0035	0.9944	1.9397	21.62	5.85	0.55	0.33
5	0.0932	0.0003	0.0078	0.9918	1.2105	**13.61**	**0.79**	0.65	0.50
10	0.0694	0.0064	0.0083	0.9852	0.8493	5.42	0.11	0.77	0.66
20	0.0543	0.0036	0.0027	0.9936	0.5756	1.19	0.05	0.92	0.84
30	0.0370	0.0001	0.0038	0.9958	0.4278	0.11	0.02	**1.00**	**0.97**
40	0.0310	0.0003	0.0058	0.9935	0.3442	0.04	0.01	1.00	1.00
50	0.0258	0.0002	0.0042	0.9950	0.2819	0.02	0.01	1.00	1.00
60	0.0203	0.0002	0.0034	0.9954	0.2321	0.01	0.00	1.00	1.00
70	0.0169	0.0001	0.0025	0.9959	0.1879	0.01	0.00	1.00	1.00
80	0.0143	0.0000	0.0025	0.9954	0.1464	0.01	0.00	1.00	1.00
90	0.0111	0.0002	0.0042	0.9920	0.1033	0.00	0.00	1.00	1.00

Table A.3. Measures of goodness-of-fit for the noon hour (12:00–13:00) obtained for different population samples of the Switzerland scenario. Values of the measures of the smallest population samples passing an acceptance threshold are marked in bold.

Sample, %	U	U_s	U_m	U_c	$RMSNE$	GEH_5, %	GEH_{10}, %	S_α^m	$S_\alpha^{m,hw}$
1	0.2610	0.0013	0.0920	0.9067	3.3671	14.24	10.58	0.67	0.21
2	0.1954	0.0026	0.0625	0.9348	2.3822	15.95	4.58	0.70	0.30
5	0.1321	0.0067	0.0506	0.9427	1.4629	**10.89**	**0.77**	0.76	0.47
10	0.1041	0.0096	0.0419	0.9485	1.0027	4.77	0.15	0.84	0.63
20	0.0810	0.0168	0.0573	0.9259	0.7208	1.38	0.08	0.93	0.81
30	0.0748	0.0264	0.0702	0.9034	0.6504	0.83	0.08	**0.97**	0.87
40	0.0719	0.0305	0.0845	0.8849	0.5754	0.64	0.07	0.99	0.91
50	0.0698	0.0324	0.0851	0.8824	0.5310	0.53	0.08	1.00	0.94
60	0.0710	0.0334	0.0887	0.8778	0.4971	0.51	0.09	1.00	**0.95**
70	0.0653	0.0394	0.0963	0.8642	0.4704	0.47	0.08	1.00	0.97
80	0.0560	0.0495	0.1056	0.8448	0.4494	0.41	0.05	1.00	0.98
90	0.0550	0.0531	0.1130	0.8338	0.4383	0.39	0.05	1.00	0.99

TABLE A.4. Measures of goodness-of-fit for the evening peak hour (17:00–18:00) obtained for different population samples of the Switzerland scenario. Values of the measures of the smallest population samples passing an acceptance threshold are marked in bold.

Sample, %	U	U_s	U_m	U_c	$RMSNE$	GEH_5, %	GEH_{10}, %	S^e_α	$S^{e,hw}_\alpha$
1	0.1856	0.0052	0.0124	0.9824	2.7455	19.23	14.49	0.52	0.24
2	0.1357	0.0061	0.0020	0.9919	1.8854	20.81	6.71	0.57	0.33
5	0.0910	0.0002	0.0204	0.9794	1.1995	**13.74**	**1.09**	0.66	0.50
10	0.0723	0.0023	0.0021	0.9955	0.8389	6.06	0.33	0.77	0.65
20	0.0622	0.0013	0.0255	0.9731	0.5827	1.76	0.29	0.91	0.82
30	0.0477	0.0042	0.0372	0.9585	0.4414	0.46	0.17	**0.99**	0.94
40	0.0445	0.0074	0.0539	0.9385	0.3632	0.37	0.17	1.00	**0.98**
50	0.0405	0.0088	0.0610	0.9300	0.3039	0.32	0.17	1.00	1.00
60	0.0356	0.0093	0.0574	0.9330	0.2603	0.29	0.15	1.00	1.00
70	0.0293	0.0114	0.0590	0.9291	0.2212	0.28	0.07	1.00	1.00
80	0.0221	0.0097	0.0457	0.9438	0.1808	0.14	0.04	1.00	1.00
90	0.0135	0.0034	0.0153	0.9791	0.1261	0.04	0.01	1.00	1.00

Table A.5. Measures of goodness-of-fit for a whole day (00:00–24:00) obtained for different population samples of the Switzerland scenario.

Sample, %	U	U_s	U_m	U_c	$RMSNE$	GEH_5, %	GEH_{10}, %	S_α^m	$S_\alpha^{m,hw}$
1	0.0782	0.0296	0.0746	0.8957	2.3386	31.14	27.59	0.22	0.19
2	0.0637	0.0439	0.0857	0.8703	1.6581	32.96	16.27	0.29	0.24
5	0.0547	0.0716	0.1398	0.7884	1.0607	25.54	8.23	0.40	0.28
10	0.0473	0.0651	0.1309	0.8038	0.7403	16.46	4.74	0.55	0.36
20	0.0451	0.0732	0.1488	0.7778	0.5306	11.84	3.90	0.68	0.40
30	0.0480	0.0935	0.1718	0.7346	0.4659	13.55	4.61	0.69	0.28
40	0.0476	0.0971	0.1810	0.7217	0.4146	13.40	4.47	0.71	0.28
50	0.0471	0.0980	0.1781	0.7237	0.3802	13.26	4.33	0.73	0.27
60	0.0468	0.0987	0.1782	0.7230	0.3539	13.14	4.26	0.73	0.28
70	0.0461	0.1026	0.1806	0.7166	0.3359	13.14	4.21	0.74	0.27
80	0.0456	0.1047	0.1776	0.7175	0.3205	13.16	4.14	0.74	0.27
90	0.0452	0.1042	0.1730	0.7227	0.3082	13.03	4.07	0.74	0.27

TABLE A.6. Measures of goodness-of-fit for public transit occupancy obtained for different population samples of the Switzerland scenario. Values of the measures of the smallest population samples passing an acceptance threshold are marked in bold.

Sample, %	U	U_s	U_m	U_c	$RMSNE$	GEH_5, %	GEH_{10}, %	S_α^{pt}
1	0.4083	0.0187	0.0048	0.9765	4.5617	23.93	9.99	0.49
2	0.2536	0.0000	0.0708	0.9291	2.0408	24.21	3.76	0.53
5	0.1650	0.0001	0.0263	0.9737	1.3663	**13.54**	**0.47**	0.65
10	0.1150	0.0001	0.0165	0.9834	1.0205	4.74	0.03	0.79
20	0.0779	0.0001	0.0066	0.9933	0.8017	0.68	0.02	0.94
30	0.0591	0.0000	0.0025	0.9974	0.6759	0.09	0.01	**1.00**
40	0.0477	0.0000	0.0025	0.9974	0.6185	0.03	0.01	1.00
50	0.0391	0.0000	0.0021	0.9978	0.5245	0.02	0.00	1.00
60	0.0323	0.0000	0.0007	0.9991	0.5216	0.02	0.00	1.00
70	0.0263	0.0000	0.0005	0.9992	0.5454	0.02	0.01	1.00
80	0.0205	0.0000	0.0005	0.9990	0.5027	0.02	0.00	1.00
90	0.0153	0.0000	0.0002	0.9990	0.5968	0.02	0.01	1.00

TABLE A.7. Performance of the simulated car transport obtained for different population samples of the Switzerland scenario.

	Morning (07:00–08:00)		Noon (12:00–13:00)		Evening (17:00–18:00)		Daily (00:00–24:00)	
Sample, %	**VHT, %**	**VKT, %**	**VHT, %**	**VKT, %**	**VHT, %**	**VKT, %**	**VHT, %**	**VKT, %**
1	123.90	95.22	131.85	99.16	132.59	94.68	133.05	100.42
2	119.19	97.38	119.43	99.08	123.52	95.56	121.47	100.11
5	106.32	99.64	102.41	99.78	106.89	101.03	101.50	100.21
10	102.58	99.61	99.31	99.80	104.92	100.60	100.20	100.02
20	98.32	100.35	95.53	100.22	97.33	103.07	93.93	100.20
30	99.51	100.37	95.32	100.21	98.32	102.41	94.68	100.21
40	99.27	100.37	95.63	100.28	97.30	102.67	94.17	100.23
50	99.14	100.28	95.54	100.22	96.81	102.64	94.00	100.17
60	99.30	100.24	95.80	100.40	96.54	102.28	94.22	100.17
70	99.24	100.08	96.72	100.29	96.71	101.77	94.78	100.06
80	99.57	100.05	97.65	100.18	98.02	101.21	96.31	100.08
90	99.80	99.99	98.95	100.19	99.09	100.52	97.98	100.00

Reference case for VHT (hours): 219 469 in the morning, 107 355 in the noon, 359 633 in the evening and 3 217 881 in a day. Reference case for VKT (vehicle-km): 12 387 429 in the morning, 6 068 409 in the noon, 16 899 257 in the evening and 168 031 450 in a day.

BIBLIOGRAPHY

1. United Nations, Department of Economic and Social Affairs, Population Division. *World Population Prospects 2019: Highlights* tech. rep. (2019).
2. United Nations, D. o. E. & Social Affairs, P. D. *World Population Prospects 2022, Online Edition* tech. rep. (2022).
3. United Nations, Department of Economic and Social Affairs, Population Division. *World Urbanization Prospects 2018: Highlights* tech. rep. (2019).
4. Schrank, D., Albert, L., Eisele, B. & Lomax, T. *Urban Mobility Report 2021* tech. rep. (2021).
5. The European Court of Auditors. *Urban mobility in EU* tech. rep. (2019).
6. Li, J., Ma, M., Xia, X. & Ren, W. The Spatial Effect of Shared Mobility on Urban Traffic Congestion: Evidence from Chinese Cities. *Sustainability* **13**, 14065 (2021).
7. KPMG International Limited. *The KPMG Survey of Sustainability Reporting 2020* tech. rep. (2020).
8. United Nations, Department of Economic and Social Affairs, Sustainable Development. *Transforming our World: The 2030 Agenda for Sustainable Development* tech. rep. (2015).
9. Anenberg, Susan and Miller, Joshua and Henze, Daven and Minjares, Ray. *A global snapshot of the air pollution-related health impacts of transportation sector emissions in 2010 and 2015* tech. rep. (2019).
10. Stansfeld, S. A. & Matheson, M. P. Noise pollution: non-auditory effects on health. *British Medical Bulletin* **68**, 243 (2003).
11. Münzel, T., Sørensen, M. & Daiber, A. Transportation noise pollution and cardiovascular disease. *Nature Reviews Cardiology* **18**, 619 (2021).
12. International Organization for Migration. *World migration report 2020* tech. rep. (2019).
13. United Nations, Department of Economic and Social Affairs, Population Division. *International Migrant Stock, Online Dataset* tech. rep. (2020).
14. Amer, M., Mustafa, A., Teller, J., Attia, S. & Reiter, S. A methodology to determine the potential of urban densification through roof stacking. *Sustainable Cities and Society* **35**, 677 (2017).
15. Ciscel, D. H. The economics of urban sprawl: Inefficiency as a core feature of metropolitan growth. *Journal of Economic Issues* **35**, 405 (2001).

16. Andong, R. F. & Sajor, E. Urban sprawl, public transport, and increasing CO2 emissions: The case of Metro Manila, Philippines. *Environment, Development and Sustainability* **19**, 99 (2017).

17. Uber Technologies, Inc. *Quarterly (1) report to US Securities and Exchange Commission* tech. rep. (2019).

18. Uber Technologies, Inc. *Annual (2019) report to US Securities and Exchange Commission* tech. rep. (2020).

19. Uber Technologies, Inc. *Annual (2021) report to US Securities and Exchange Commission* tech. rep. (2022).

20. Wiering, M. A. *Multi-agent reinforcement learning for traffic light control* in *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)* (2000), 1151.

21. Speranza, M. G. Trends in transportation and logistics. *European Journal of Operational Research* **264**, 830 (2018).

22. Litman, T. The new transportation planning paradigm. *Institute of Transportation Engineers. ITE Journal* **83**, 20 (2013).

23. Burns, L. D. Sustainable mobility: a vision of our transport future. *Nature* **497**, 181 (2013).

24. Hars, A. in *Marktplätze im Umbruch* 539 (Springer, 2015).

25. Andrew, R. M. A comparison of estimates of global carbon dioxide emissions from fossil carbon sources. *Earth System Science Data* **12**, 1437 (2020).

26. International Panel on Climate Change. *Climate Change 2022: Mitigation of Climate Change: Summary for Policymakers* tech. rep. (2022).

27. Huang, Y., Li, S. & Qian, Z. S. Optimal deployment of alternative fueling stations on transportation networks considering deviation paths. *Networks and Spatial Economics* **15**, 183 (2015).

28. Dijk, M., Orsato, R. J. & Kemp, R. The emergence of an electric mobility trajectory. *Energy Policy* **52**, 135 (2013).

29. Childress, S., Nichols, B., Charlton, B. & Coe, S. Using an activity-based model to explore the potential impacts of automated vehicles. *Transportation Research Record* **2493**, 99 (2015).

30. IEA. *Global EV Outlook 2022* tech. rep. (2022).

31. Biresselioglu, M. E., Kaplan, M. D. & Yilmaz, B. K. Electric mobility in Europe: A comprehensive review of motivators and barriers in decision making processes. *Transportation Research Part A: Policy and Practice* **109**, 1 (2018).

32. Prettenthaler, F. E. & Steininger, K. W. From ownership to service use lifestyle: the potential of car sharing. *Ecological Economics* **28**, 443 (1999).

33. Cervero, R., Golub, A. & Nee, B. City CarShare: longer-term travel demand and car ownership impacts. *Transportation Research Record* **1992**, 70 (2007).

34. Plagowski, P., Saprykin, A., Chokani, N. & Shokrollah-Abhari, R. Impact of electric vehicle charging–An agent-based approach. *IET Generation, Transmission & Distribution* **15**, 2605 (2021).

35. Guille, C. & Gross, G. A conceptual framework for the vehicle-to-grid (V2G) implementation. *Energy Policy* **37**, 4379 (2009).

36. Jittrapirom, P., Caiati, V., Feneri, A.-M., Ebrahimigharehbaghi, S., González, M. J. A., Narayan, J., *et al.* Mobility as a Service: A Critical Review of Definitions, Assessments of Schemes, and Key Challenges. *Urban Planning* **2**, 13 (2017).

37. Karlsson, I. M., Sochor, J. & Strömberg, H. Developing the 'Service' in Mobility as a Service: experiences from a field trial of an innovative travel brokerage. *Transportation Research Procedia* **14**, 3265 (2016).

38. Audouin, M. & Finger, M. The development of Mobility-as-a-Service in the Helsinki metropolitan area: A multi-level governance analysis. *Research in Transportation Business & Management* **27**, 24 (2018).

39. Gerlough, D. L. *Simulation of freeway traffic on a general-purpose discrete variable computer* PhD thesis (University of California, Los Angeles, 1955).

40. Goode, H. H., Pollmar, C. H. & Wright, J. B. *The use of a digital computer to model a signalized intersection* tech. rep. (1956).

41. Oppenheim, N. *Urban Travel Demand Modeling: From Individual Choices to General Equilibrium* (Wiley-Interscience Publication, New York, NY, USA, 1995).

42. Kitamura, R., Pas, E. I., Lula, C. V., Lawton, T. K. & Benson, P. E. The sequenced activity mobility simulator (SAMS): an integrated approach to modeling transportation, land use and air quality. *Transportation* **23**, 267 (1996).

43. Cervero, R. Alternative approaches to modeling the travel-demand impacts of smart growth. *Journal of the American Planning Association* **72**, 285 (2006).

44. Bonabeau, E. Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* **99**, 7280 (2002).

45. Zhang, L. & Levinson, D. Agent-based approach to travel demand modeling: exploratory analysis. *Transportation Research Record* **1898**, 28 (2004).

46. Eppstein, M. J., Grover, D. K., Marshall, J. S. & Rizzo, D. M. An agent-based model to study market penetration of plug-in hybrid electric vehicles. *Energy Policy* **39**, 3789 (2011).

47. Fagnant, D. J. & Kockelman, K. M. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies* **40**, 1 (2014).

48. Zhuge, C. & Shao, C. Agent-based modelling of locating public transport facilities for conventional and electric vehicles. *Networks and Spatial Economics* **18**, 875 (2018).

49. Abar, S., Theodoropoulos, G. K., Lemarinier, P. & O'Hare, G. M. Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review* **24**, 13 (2017).

50. Barceló, J., Ferrer, J., García, D. & Grau, R. in *Equilibrium and Advanced Transportation Modelling* 1 (Boston, MA, USA, 1998).

51. Aydt, H., Xu, Y., Lees, M. & Knoll, A. *A multi-threaded execution model for the agent-based SEMSim traffic simulation* in *Communications in Computer and Information Science* **402** (Springer, Berlin, Heidelberg, 2013), 1.

52. Cameron, G. D. & Duncan, G. I. PARAMICS - Parallel microscopic simulation of road traffic. *Journal of Supercomputing* **10**, 25 (1996).

53. Rickert, M. & Nagel, K. Dynamic traffic assignment on parallel computers in TRANSIMS. *Future Generation Computer Systems* **17**, 637 (2001).

54. Cetin, N., Burri, A. & Nagel, K. A large-scale agent-based traffic microsimulation based on queue model. *In Proceedings of Swiss Transport Research Conference (STRC), Monte Verita, CH* (2003).

55. Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R. & Lavanya, R. *A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation* in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (2005), 813.

56. Turek, W. Erlang-based desynchronized urban traffic simulation for high-performance computing systems. *Future Generation Computer Systems* **79**, 645 (2018).

57. Nökel, K. & Schmidt, M. Parallel DYNEMO: Meso-Scopic Traffic Flow Simulation on Large Networks. *Networks and Spatial Economics* **2**, 387 (2002).

58. Kiesling, T. & Luthi, J. *Towards time-parallel road traffic simulation* in *Workshop on Principles of Advanced and Distributed Simulation (PADS'05)* (2005), 7.

59. Xu, Y. & Tan, G. *An offline road network partitioning solution in distributed transportation simulation* in *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications* (IEEE Computer Society, Dublin, Ireland, 2012), 210.

60. Zhang, D., Jiang, C. & Li, S. A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory* **17**, 1032 (2009).

61. Namekawa, M., Satoh, A., Mori, H., Yikai, K. & Nakanishi, T. Clock synchronization algorithm for parallel road-traffic simulation system in a wide area. *Mathematics and Computers in Simulation* **48**, 351 (1999).

62. Suzumura, T. & Kanezashi, H. *Multi-modal traffic simulation platform on parallel and distributed systems* in *Proceedings of the Winter Simulation Conference 2014* (2014), 769.

63. Suzumura, T., McArdle, G. & Kanezashi, H. *A high performance multi-modal traffic simulation platform and its case study with the Dublin city* in *Proceedings of the 2015 Winter Simulation Conference* (IEEE Press, Huntington Beach, California, 2015), 767.

64. Goldberg, A. V. & Harrelson, C. *Computing the Shortest Path : A* Search Meets Graph Theory* in *SODA '05 Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (2005), 156.

65. Sanders, P. & Schultes, D. *Highway hierarchies hasten exact shortest path queries* in (Springer Berlin Heidelberg, 2005), 568.

66. Graur, D., Bruno, R., Bischoff, J., Rieser, M., Scherr, W., Hoefler, T. & Alonso, G. Hermes: Enabling efficient large-scale simulation in MATSim. *Procedia Computer Science* **184**, 635 (2021).

67. Moore, G. E. Cramming more components onto integrated circuits. *Electronics* **38**, 114 (1965).

68. Moore, G. E. *Progress in digital integrated electronics* in *Electron Devices Meeting* **21** (1975), 11.

69. Rupp, K. *Microprocessor Trend Data* `https://github.com/karlrupp/microprocessor-trend-data` [Accessed: 31-Mar-2022]. 2022.

70. Amdahl, G. M. *Validity of the single processor approach to achieving large scale computing capabilities* in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (1967), 483.

71. Rupp, K. *CPU-GPU-MIC Comparison Charts* `https://github.com/karlrupp/cpu-gpu-mic-comparison` [Accessed: 31-Mar-2022]. 20219.

72. TOP500. *TOP500 List* `https://www.top500.org/` [Accessed: 31-Mar-2022]. 2022.

73. Rajamanickam, S. *Towards simulations on the Exascale hardware and beyond.* tech. rep. (Sandia National Laboratory (SNL-NM), Albuquerque, NM (United States), 2020).

74. MarketsandMarkets. *Cloud Computing Market by Service, Deployment Model, Organization Size, Vertical And Region – Global Forecast to 2026* `https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html` [Accessed: 31-Mar-2022]. 2021.

75. Strippgen, D. & Nagel, K. *Multi-agent traffic simulation with CUDA* in *Proceedings of the 2009 International Conference on High Performance Computing and Simulation, HPCS 2009* (2009), 106.

76. Horni, A., Nagel, K. & Axhausen, K. W. *The multi-agent transport simulation MATSim* (Ubiquity Press London, 2016).

77. Perumalla, K. S., Aaby, B. G., Yoginath, S. B. & Seal, S. K. *GPU-based real-time execution of vehicular mobility models in large-scale road network scenarios* in *2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation* (2009), 95.

78. Shen, Z., Wang, K. & Zhu, F. *Agent-based traffic simulation and traffic signal timing optimization with GPU* in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (2011), 145.

79. Wang, K. & Shen, Z. *A GPU based TrafficParallel Simulation Module of Artificial Transportation Systems* in *Proceedings of 2012 IEEE International Conference on Service Operations and Logistics, and Informatics* (2012), 160.

80. Hirabayashi, M., Kato, S., Edahiro, M. & Sugiyama, Y. Toward GPU-accelerated traffic simulation and its real-time challenge (2012).

81. Bando, M., Hasebe, K., Nakanishi, K., Nakayama, A., Shibata, A. & Sugiyama, Y. Phenomenological study of dynamical model of traffic flow. *Journal de Physique I* **5**, 1389 (1995).

82. Sano, Y. & Fukuta, N. *A GPU-based framework for large-scale multi-agent traffic simulations* in *2013 Second IIAI International Conference on Advanced Applied Informatics* (2013), 262.

83. Sano, Y. & Fukuta, N. A GPU-Based Programming Framework for Highly-Scalable Multi-Agent Traffic Simulations. *Journal of Advanced Computational Intelligence and Intelligent Informatics* **18**, 581 (2014).

84. Sano, Y., Kadono, Y. & Fukuta, N. in *Recent Advances in Agent-based Complex Automated Negotiation* 141 (Springer, 2016).

85. Xu, Y., Song, X., Weng, Z. & Tan, G. An Entry Time-based Supply Framework (ETSF) for mesoscopic traffic simulations. *Simulation Modelling Practice and Theory* **47**, 182 (2014).

86. Xu, Y., Tan, G., Li, X. & Song, X. *Mesoscopic Traffic Simulation on CPU/GPU* in *Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* (2014), 39.

87. Song, X., Xie, Z., Xu, Y., Tan, G., Tang, W., Bi, J. & Li, X. Supporting real-world network-oriented mesoscopic traffic simulation on GPU. *Simulation Modelling Practice and Theory* **74**, 46 (2017).

88. Vu, V. A. & Tan, G. *High-performance mesoscopic traffic simulation with GPU for large scale networks* in *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2017), 1.

89. Vu, V. A. & Tan, G. A framework for mesoscopic traffic simulation in GPU. *IEEE Transactions on Parallel and Distributed Systems* **30**, 1691 (2019).

90. Heywood, P., Richmond, P. & Maddock, S. *Road network simulation using FLAME GPU* 430 (Springer, Cham, 2015).

91. Richmond, P., Walker, D., Coakley, S. & Romano, D. High performance cellular level agent-based simulation with FLAME for the GPU. *Briefings in Bioinformatics* **11**, 334 (2010).

92. Heywood, P., Maddock, S., Casas, J., Garcia, D., Brackstone, M. & Richmond, P. Data-parallel agent-based microscopic road network simulation using graphics processing units. *Simulation Modelling Practice and Theory* **83**, 188 (2018).

93. Barceló, J. & Casas, J. in *Simulation approaches in transportation analysis* 57 (Springer, 2005).

94. Yedavalli, P., Kumar, K. & Waddell, P. Microsimulation analysis for network traffic assignment (MANTA) at metropolitan-scale for agile transportation planning. *Transportmetrica A: Transport Science*, 1 (2021).

95. Xiao, J., Andelfinger, P., Eckhoff, D., Cai, W. & Knoll, A. A survey on agent-based simulation using hardware accelerators. *ACM Computing Surveys (CSUR)* **51**, 1 (2019).

96. Xiao, J., Andelfinger, P., Eckhoff, D., Cai, W. & Knoll, A. *Exploring execution schemes for agent-based traffic simulation on heterogeneous hardware* in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2018), 1.

97. Xiao, J., Kilinç, G., Andelfinger, P., Eckhoff, D., Cai, W. & Knoll, A. *Pedal to the Bare Metal: Road Traffic Simulation on FPGAs Using High-Level Synthesis* in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation* (2020), 117.

98. Rajf, D. & Potuzak, T. *Comparison of road traffic simulation speed on CPU and GPU* in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* (2019), 1.

99. Tripp, J. L., Mortveit, H. S., Hansson, A. A. & Gokhale, M. *Metropolitan road traffic simulation on FPGAs* in *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM05)* (2005), 117.

100. Fellendorf, M. *VISSIM: A microscopic simulation tool to evaluate actuated signal control including bus priority* in *64th Institute of Transportation Engineers Annual Meeting* **32** (1994), 1.

101. Smith, L., Beckman, R., Anson, D., Nagel, K. & Williams, M. E. *TRANSIMS: Transportation analysis and simulation system* tech. rep. (1995).

102. Behrisch, M., Bieker, L., Erdmann, J. & Krajzewicz, D. *SUMO – Simulation of Urban MObility* in *SIMUL 2011: The Third International Conference on Advances in System Simulation* (2011), 55.

103. Ruiz-Rosero, J., Ramirez-Gonzalez, G. & Khanna, R. Masivo: Parallel Simulation Model Based on OpenCL for Massive Public Transportation Systems' Routes. *Electronics* **8**, 1501 (2019).

104. Bischoff, J. & Maciejewski, M. Simulation of city-wide replacement of private cars with autonomous taxis in Berlin. *Procedia Computer Science* **83**, 237 (2016).

105. Maciejewski, M. & Bischoff, J. Congestion effects of autonomous taxi fleets. *Transport* **33**, 971 (2018).

106. Levin, M. W., Kockelman, K. M., Boyles, S. D. & Li, T. A general framework for modeling shared autonomous vehicles with dynamic network-loading and dynamic ride-sharing application. *Computers, Environment and Urban Systems* **64**, 373 (2017).

107. Hörl, S., Ruch, C., Becker, F., Frazzoli, E. & Axhausen, K. W. Fleet operational policies for automated mobility: A simulation assessment for Zurich. *Transportation Research Part C: Emerging Technologies* **102**, 20 (2019).

108. Ben-Dor, G., Dmitrieva, B., Maciejewski, M., Bischoff, J., Ben-Elia, E. & Benenson, I. *MATSim simulations in the Tel Aviv Metropolitan Area: Direct competition between public transport and cars on the same roadway* in *hEART 2017: 6th Symposium of the European Association for Research in Transportation* (2017).

109. Bischoff, J. & Maciejewski, M. *Autonomous Taxicabs in Berlin – A Spatiotemporal Analysis of Service Performance* in *Transportation Research Procedia* **19** (Elsevier, 2016), 176.

110. Simoni, M. D., Pel, A. J., Waraich, R. A. & Hoogendoorn, S. P. Marginal cost congestion pricing based on the network fundamental diagram. *Transportation Research Part C: Emerging Technologies* **56**, 221 (2015).

111. Erath, A., Fourie, P., van Eggermond, M., Ordoñez, S., Chakirov, A. & Axhausen, K. W. *Large-scale agent-based transport demand model for Singapore* in *13th International Conference on Travel Behaviour Research* **790** (Future Cities Laboratory (FCL), 2012).

112. Bösch, P., Ciari, F. & Axhausen, K. The IVT 2015 baseline scenario. *16th Swiss Transport Research Conference* (2016).

113. Kwak, M., Arentze, T., de Romph, E. & Rasouli, S. *Activity-based dynamic traffic modeling: Influence of population sampling fraction size on simulation error* in *Proceedings International Association of Travel Behavior Research Conference* (2012).

114. Llorca, C. & Moeckel, R. Effects of scaling down the population for agent-based traffic simulations. *Procedia Computer Science* **151**, 782 (2019).

115. Ben-Dor, G., Ben-Elia, E. & Benenson, I. Spatiotemporal Implications of Population Downscaling: A MATSim Study of Sioux Falls Morning Peak Traffic. *Procedia Computer Science* **170**, 720 (2020).

116. Ben-Dor, G., Ben-Elia, E. & Benenson, I. Population downscaling in multi-agent transportation simulations: A review and case study. *Simulation Modelling Practice and Theory* **108**, 102233 (2021).

117. Sewall, J., Wilkie, D., Merrell, P. & Lin, M. C. *Continuum traffic simulation* in *Computer Graphics Forum* **29** (2010), 439.

118. Sewall, J., Van Den Berg, J., Lin, M. & Manocha, D. Virtualized Traffic: Reconstructing Traffic Flows from Discrete Spatiotemporal Data. *IEEE transactions on visualization and computer graphics* **17**, 26 (2010).

119. Sewall, J., Wilkie, D. & Lin, M. C. *Interactive hybrid simulation of large-scale traffic* in *Proceedings of the 2011 SIGGRAPH Asia Conference* (2011), 1.

120. Suzumura, T., Kato, S., Imamichi, T., Takeuchi, M., Kanezashi, H., Ide, T. & Onodera, T. *X10-based massive parallel large-scale traffic flow simulation* in *Proceedings of the 2012 ACM SIGPLAN X10 Workshop* (2012), 1.

121. Shen, J. & Jin, X. Detailed traffic animation for urban road networks. *Graphical Models* **74**, 265 (2012).

122. Lu, W., Liu, C., Thomas, N., Bhaduri, B. L. & Han, L. D. Global system for transportation simulation and visualization in emergency evacuation scenarios. *Transportation Research Record* **2529**, 46 (2015).

123. Dobson, J. E., Bright, E. A., Coleman, P. R., Durfee, R. C. & Worley, B. A. LandScan: A Global Population Database for Estimating Populations at Risk. *Photogrammetric Engineering and Remote Sensing* **66**, 849 (2000).

124. Gehlot, H., Zhan, X., Qian, X., Thompson, C., Kulkarni, M. & Ukkusuri, S. V. A-RESCUE 2.0: A High-Fidelity, Parallel, Agent-Based Evacuation Simulator. *Journal of Computing in Civil Engineering* **33**, 04018059 (2019).

125. Kim, S.-S., Min, O. & Kim, Y.-K. *SALT-Viz: Real-Time Visualization for Large-Scale Traffic Simulation* in *2019 IEEE Transportation Electrification Conference and Expo, Asia-Pacific (ITEC Asia-Pacific)* (2019), 1.

126. Charlton, B. & Laudan, J. Web-Based Data Visualization Platform for MATSim. *Transportation Research Record* **2674**, 124 (2020).

127. Saprykin, A., Chokani, N. & Abhari, R. S. *Gridlock resolution in a GPU-accelerated traffic queue model* in. **170** (Elsevier, 2020), 681.

128. Sung, I. J., Liu, G. D. & Hwu, W. M. W. *DL: A data layout transformation system for heterogeneous computing* in *2012 Innovative Parallel Computing (InPar)* (2012), 1.

129. Mei, G. & Tian, H. Impact of data layouts on the efficiency of GPU-accelerated IDW interpolation. *SpringerPlus* **5** (2016).

130. Eser, P., Singh, A., Chokani, N. & Abhari, R. S. Effect of increased renewables generation on operation of thermal power plants. *Applied Energy* **164**, 723 (2016).

131. Marini, M., Gawlikowska, A. P., Rossi, A., Chokani, N., Klumpner, H. & Abhari, R. S. The impact of future cities on commuting patterns: An agent-based approach. *Environment and Planning B: Urban Analytics and City Science* **0**, 1 (2018).

132. Pagani, M., Korosec, W., Chokani, N. & Abhari, R. S. User behaviour and electric vehicle charging infrastructure: An agent-based model assessment. *Applied Energy* **254**, 113680 (2019).

133. Holland, J. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence* (University of Michigan Press, Oxford, UK, 1975).

134. Nash, J. Non-Cooperative Games. *The Annals of Mathematics* **54**, 286 (1951).

135. Wardrop, J. G. Some Theoretical Aspects of Road Traffic Research. *Proceedings of the Institution of Civil Engineers* **1**, 325 (1952).

136. Charypar, D. & Nagel, K. Generating complete all-day activity plans with genetic algorithms. *Transportation* **32**, 369 (2005).

137. Vickrey, W. Congestion Theory and Transport Investment. *The American Economic Review* **59**, 251 (1969).

138. Joubert, C. J., Saprykin, A., Chokani, N. & Abhari, R. S. Large-scale agent-based modelling of street robbery using graphical processing units and reinforcement learning. *Computers, Environment and Urban Systems* **94**, 101757 (2022).

139. Waraich, R. A., Charypar, D., Balmer, M. & Axhausen, K. W. in *Computational Approaches for Urban Environments* 211 (Springer, Cham, 2015).

140. Nagel, K. & Barrett, C. L. Using microsimulation feedback for trip adaptation for realistic traffic in Dallas. *International Journal of Modern Physics C* **8**, 505 (1997).

141. Nagel, K. & Rickert, M. *Issues of Simulation-based Route Assignment* tech. rep. (Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 1999).

142. Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D. & Werneck, R. F. in *Algorithm Engineering* 19 (Springer, 2016).

143. Dijkstra, E. W. *et al.* A note on two problems in connexion with graphs. *Numerische Mathematik* **1**, 269 (1959).

144. Bleiweiss, A. *GPU accelerated pathfinding* in *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2008), 65.

145. Djidjev, H., Thulasidasan, S., Chapuis, G., Andonov, R. & Lavenier, D. *Efficient multi-GPU computation of all-pairs shortest paths* in *2014 IEEE 28th International Parallel and Distributed Processing Symposium* (2014), 360.

146. Zhou, Y. & Zeng, J. *Massively parallel A* search on a GPU* in *Proceedings of the AAAI Conference on Artificial Intelligence* **29** (2015).

147. Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A. & Owens, J. D. *Gunrock: A high-performance graph processing library on the GPU* in *Proceedings of the 21st ACM SIGPLAN symposium on principles and practice of parallel programming* (2016), 1.

148. He, X., Yao, Y., Chen, Z., Sun, J. & Chen, H. Efficient parallel A* search on multi-GPU system. *Future Generation Computer Systems* **123**, 35 (2021).

149. Haklay, M. & Weber, P. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive computing* **7**, 12 (2008).

150. Goldberg, A. V. & Werneck, R. F. F. *Computing Point-to-Point Shortest Paths from External Memory* in *ALENEX/ANALCO* (2005), 26.

151. Johnson, D. B. Priority queues with update and finding minimum spanning trees. *Information Processing Letters* **4**, 53 (1975).

152. Jung, H. The d-deap*: A fast and simple cache-aligned d-ary deap. *Information Processing Letters* **93**, 63 (2005).

153. Liu, R. in *Fundamentals of Traffic Simulation* 295 (Springer, New York, NY, USA, 2010).

154. Ben-Akiva, M., Bierlaire, M., Koutsopoulos, H. & Mishalani, R. *DynaMIT: a simulation-based system for traffic prediction* in *DACCORD Short Term Forecasting Workshop* (1998).

155. Barceló, J. in *Fundamentals of Traffic Simulation* 1 (Springer, New York, NY, USA, 2010).

156. Ni, D. *2DSIM: a prototype of nanoscopic traffic simulation* in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683)* (2003), 47.

157. Dia, H. & Panwai, S. *Nanoscopic traffic simulation: enhanced models of driver behaviour for ITS and telematics simulations* in *International Symposium on Transport Simulation, 8th, 2008, Surfers Paradise, Queensland, Australia* (2008).

158. Xu, Y. *Investigate Entry-Time Based Supply Framework on GPU* `https://github.com/xuyannus/traffic_simulation_on_gpu` [Accessed: 11-Oct-2018]. 2014.

159. Zhou, H., Dorsman, J., Snelder, M., de Romph, E. & Mandjes, M. GPU-based parallel computing for activity-based travel demand models. *Procedia Computer Science* **151**, 726 (2019).

160. Sheppard, C., Waraich, R., Campbell, A., Pozdnukov, A. & Gopal, A. R. *Modeling plug-in electric vehicle charging demand with BEAM: The framework for behavior energy autonomy mobility* tech. rep. (Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA (United States), 2017).

161. Zhang, H. M., Nie, Y. (& Qian, Z. (Modelling network flow with and without link interactions: The cases of point queue, spatial queue and cell transmission model. *Transportmetrica B* **1**, 33 (2013).

162. Jeff Ban, X., Pang, J. S., Liu, H. X. & Ma, R. Continuous-time point-queue models in dynamic network loading. *Transportation Research Part B: Methodological* **46**, 360 (2012).

163. Zhou, X. & Taylor, J. DTAlite: A queue-based mesoscopic traffic simulator for fast model evaluation and calibration. *Cogent Engineering* **1** (2014).

164. Daganzo, C. F. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B* **28**, 269 (1994).

165. Daganzo, C. F. The cell transmission model, part II: Network traffic. *Transportation Research Part B* **29**, 79 (1995).

166. Yperman, I. *The Link Transmission Model for dynamic network loading* PhD thesis (Katholieke Universiteit Leuven, Leuven, Belgium, 2007).

167. Gawron, C. *Simulation-based traffic assignment: computing user equilibria in large street networks* PhD thesis (University of Cologne, Cologne, Germany, 1998).

168. Hausknecht, M., Au, T.-C., Stone, P., Fajardo, D. & Waller, T. *Dynamic lane reversal in traffic management* in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)* (2011), 1929.

169. Chu, K. F., Lam, A. Y. & Li, V. O. *Dynamic lane reversal routing and scheduling for connected autonomous vehicles* in *2017 International Smart Cities Conference (ISC2)* (2017), 1.

170. Rämä, P. Effects of weather-controlled variable speed limits and warning signs on driver behavior. *Transportation Research Record* **1689**, 53 (1999).

171. Lee, C., Hellinga, B. & Saccomanno, F. Evaluation of variable speed limits to improve traffic safety. *Transportation Research Part C: Emerging Technologies* **14**, 213 (2006).

172. Federal Statistical Office / Federal Office for Spatial Development. *Mobility and Transport Microcensus (MTMC) 2010* (Bern, Switzerland, 2012).

173. Marini, M., Brunner, C., Chokani, N. & Abhari, R. S. Enhancing response preparedness to influenza epidemics: Agent-based study of 2050 influenza season in Switzerland. *Simulation Modelling Practice and Theory* **103**, 102091 (2020).

174. Federal Statistical Office. *Business and Enterprise Register (2013)* (Neuchâtel, Switzerland, 2013).

175. Federal Statistical Office. *Statistics on Enterprise Structure (STATENT) (2012)* (Neuchâtel, Switzerland, 2012).

176. Swiss Open Data Platform. *Timetable 2018 (GTFS)* `https://opentransportdata.swiss/en/dataset/timetable-2018-gtfs` [Accessed: 07-Jul-2022].

177. Federal Roads Office. *Schweizerische automatische Strassenverkehrszählung (SASVZ)* `https://www.bfs.admin.ch/asset/de/dz-b-11.03.01-AVZ-2012z` [Accessed: 07-Jul-2022].

178. Saprykin, A., Chokani, N. & Abhari, R. S. GEMSim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios. *Simulation Modelling Practice and Theory* **94**, 199 (2019).

179. Daganzo, C. F. Urban gridlock: Macroscopic modeling and mitigation approaches. *Transportation Research Part B: Methodological* **41**, 49 (2007).

180. Daganzo, C. F. & Geroliminis, N. An analytical approximation for the macroscopic fundamental diagram of urban traffic. *Transportation Research Part B: Methodological* **42**, 771 (2008).

181. Geroliminis, N. & Sun, J. Hysteresis phenomena of a macroscopic fundamental diagram in freeway networks. *Procedia Social and Behavioral Sciences* **17**, 213 (2011).

182. Kerner, B. S. Physics of traffic gridlock in a city. *Physical Review E* **84**, 045102 (2011).

183. Rieser, M. & Nagel, K. Network breakdown "at the edge of chaos" in multi-agent traffic simulations. *The European Physical Journal B* **63**, 321 (2008).

184. Rickert, M. & Nagel, K. Experiences with a simplified microsimulation for the Dallas/Fort-Worth area. *International Journal of Modern Physics C* **8**, 483 (1997).

185. Horni, A. & Axhausen, K. W. *Gridlock modeling with MATSim* in *14th Swiss Transport Research Conference (STRC 2014)* (2014).

186. Olstam, J. J. & Tapani, A. *Comparison of car-following models* (Swedish National Road and Transport Research Institute Linköping, Sweden, 2004).

187. Gazis, D. C., Herman, R. & Potts, R. B. Car-following theory of steady-state traffic flow. *Operations Research* **7**, 499 (1959).

188. Pipes, Louis A. An operational analysis of traffic dynamics. *Journal of Applied Physics* **24**, 274 (1953).

189. Kometani, E. & Sasaki, T. A safety index for traffic with linear spacing. *Operations Research* **7**, 704 (1959).

190. Gipps, P. G. A behavioural car-following model for computer simulation. *Transportation Research Part B: Methodological* **15**, 105 (1981).

191. Brackstone, M. & McDonald, M. Car-following: a historical review. *Transportation Research Part F: Traffic Psychology and Behaviour* **2**, 181 (1999).

192. Wiedemann, R. Simulation des Strassenverkehrsflusses (In German) (1974).

193. Wiedemann, R. *Modelling of RTI-Elements on multi-lane roads* in *Drive Conference (1991: Brussels, Belgium)* **2** (1991).

194. Fritzsche, Hans-Thomas. A model for traffic simulation. *Traffic Engineering & Control* **35** (1994).

195. Al-Shihabi, T. & Mourant, R. R. Toward more realistic driving behavior models for autonomous vehicles in driving simulators. *Transportation Research Record* **1843**, 41 (2003).

196. Kesting, A. & Treiber, M. Calibrating car-following models by using trajectory data: Methodological study. *Transportation Research Record* **2088**, 148 (2008).

197. Ossen, S. & Hoogendoorn, S. P. Heterogeneity in car-following behavior: Theory and empirics. *Transportation Research Part C: Emerging Technologies* **19**, 182 (2011).

198. Sato, T., Akamatsu, M., Zheng, P. & McDonald, M. *Comparison of car following behavior between UK and Japan* in *2009 ICCAS-SICE* (2009), 4155.

199. Proctor, C. L., Grimes, W. D., Fournier Jr, D. J., Rigol Jr, J. & Sunseri, M. G. Analysis of acceleration in passenger cars and heavy trucks. *SAE transactions*, 283 (1995).

200. Hoberock, L. L. *A survey of longitudinal acceleration comfort studies in ground transportation vehicles* tech. rep. (Council for Advanced Transportation Studies, 1976).

201. Panwai, S. & Dia, H. Comparative evaluation of microscopic car-following behavior. *IEEE Transactions on Intelligent Transportation Systems* **6**, 314 (2005).

202. Wee, S. H., Cheu, R. L., Lee, D.-H. & Chan, W. T. in *Applications of Advanced Technologies in Transportation (2002)* 64 (2002).

203. Fiori, C., Ahn, K. & Rakha, H. A. Power-based electric vehicle energy consumption model: Model development and validation. *Applied Energy* **168**, 257 (2016).

204. Stone, J. E., Gohara, D. & Shi, G. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science & Engineering* **12**, 66 (2010).

205. Jo, G., Jeon, W. J., Jung, W., Taft, G. & Lee, J. *OpenCL framework for ARM processors with NEON support* in *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing* (2014), 33.

206. Winterstein, F. & Constantinides, G. *Pass a pointer: Exploring shared virtual memory abstractions in OpenCL tools for FPGAs* in *2017 International Conference on Field Programmable Technology (ICFPT)* (2017), 104.

207. Komatsu, K., Sato, K., Arai, Y., Koyama, K., Takizawa, H. & Kobayashi, H. *Evaluating performance and portability of OpenCL programs* in *The fifth International Workshop on Automatic Performance Tuning* **66** (2010), 1.

208. Matsumoto, K., Nakasato, N. & Sedukhin, S. G. *Performance tuning of matrix multiplication in OpenCL on different GPUs and CPUs* in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis* (2012), 396.

209. Zhang, Y., Sinclair, M. & Chien, A. A. *Improving performance portability in OpenCL programs* in *International Supercomputing Conference* (2013), 136.

210. Li, Y., Zhang, Y.-Q., Liu, Y.-Q., Long, G.-P. & Jia, H.-P. MPFFT: An auto-tuning FFT library for OpenCL GPUs. *Journal of Computer Science and Technology* **28**, 90 (2013).

211. Nugteren, C. & Codreanu, V. *CLTune: A generic auto-tuner for OpenCL kernels* in *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip* (2015), 195.

212. Wang, D. *Meeting green computing challenges* in *2008 10th Electronics Packaging Technology Conference* (2008), 121.

213. Harmon, R. R. & Auseklis, N. *Sustainable IT services: Assessing the impact of green computing practices* in *PICMET'09-2009 Portland International Conference on Management of Engineering & Technology* (2009), 1707.

214. Liu, Liang and Wang, Hao and Liu, Xue and Jin, Xing and He, Wen Bo and Wang, Qing Bo and Chen, Ying. *GreenCloud: a new architecture for green data center* in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session* (2009), 29.

215. Jain, A., Mishra, M., Peddoju, S. K. & Jain, N. *Energy efficient computing-green cloud computing* in *2013 International Conference on Energy Efficient Technologies for Sustainability* (2013), 978.

216. Feng, W.-c. & Cameron, K. The Green500 List: Encouraging Sustainable Supercomputing. *Computer* **40**, 50 (2007).

217. Dong, T., Dobrev, V., Kolev, T., Rieben, R., Tomov, S. & Dongarra, J. *A step towards energy efficient computing: Redesigning a hydrodynamic application on CPU-GPU* in *2014 IEEE 28th International Parallel and Distributed Processing Symposium* (2014), 972.

218. Lukarski, D. & Skoglund, T. *A priori power estimation of linear solvers on multi-core processors* in *UPMARC Winter Meeting* (2013).

219. Too, L. & Earl, G. Public transport service quality and sustainable development: A community stakeholder perspective. *Sustainable Development* **18**, 51 (2010).

220. Redman, L., Friman, M., Gärling, T. & Hartig, T. Quality attributes of public transport that attract car users: A research review. *Transport policy* **25**, 119 (2013).

221. Miller, P., de Barros, A. G., Kattan, L. & Wirasinghe, S. Public transportation and sustainability: A review. *KSCE Journal of Civil Engineering* **20**, 1076 (2016).

222. Litman, T. *Evaluating public transit benefits and costs* (Victoria Transport Policy Institute Victoria, BC, Canada, 2015).

223. Truong, L. T. & Currie, G. Macroscopic road safety impacts of public transport: A case study of Melbourne, Australia. *Accident Analysis & Prevention* **132**, 105270 (2019).

224. Gray, D., Shaw, J. & Farrington, J. Community transport, social capital and social exclusion in rural areas. *Area* **38**, 89 (2006).

225. Schwanen, T., Lucas, K., Akyelken, N., Solsona, D. C., Carrasco, J.-A. & Neutens, T. Rethinking the links between social exclusion and transport disadvantage through the lens of social capital. *Transportation Research Part A: Policy and Practice* **74**, 123 (2015).

226. Geurs, K. T., de Bok, M. & Zondag, B. in *Accessibility analysis and transport planning* (Edward Elgar Publishing, 2012).

227. Zhao, P. Sustainable urban expansion and transportation in a growing megacity: Consequences of urban sprawl for mobility on the urban fringe of Beijing. *Habitat International* **34**, 236 (2010).

228. De Vos, J. & Witlox, F. Transportation policy as spatial planning tool; reducing urban sprawl by increasing travel costs and clustering infrastructure and public transportation. *Journal of Transport Geography* **33**, 117 (2013).

229. Wen, L. M. & Rissel, C. Inverse associations between cycling to work, public transport, and overweight and obesity: findings from a population based study in Australia. *Preventive medicine* **46**, 29 (2008).

230. Litman, T. *Evaluating public transportation health benefits* (Victoria Transport Policy Institute, 2012).

231. Rojas-Rueda, D., de Nazelle, A., Teixidó, O. & Nieuwenhuijsen, M. J. Replacing car trips by increasing bike and public transport in the greater Barcelona metropolitan area: a health impact assessment study. *Environment International* **49**, 100 (2012).

232. Patterson, R., Webb, E., Millett, C. & Laverty, A. Physical activity accrued as part of public transport use in England. *Journal of Public Health* **41**, 222 (2019).

233. Wadud, Z. Fully automated vehicles: A cost of ownership analysis to inform early adoption. *Transportation Research Part A: Policy and Practice* **101**, 163 (2017).

234. Bösch, P. M., Becker, F., Becker, H. & Axhausen, K. W. Cost-based analysis of autonomous mobility services. *Transport Policy* **64**, 76 (2018).

235. Scheltes, A. & de Almeida Correia, G. H. Exploring the use of automated vehicles as last mile connection of train trips through an agent-based simulation model: An application to Delft, Netherlands. *International Journal of Transportation Science and Technology* **6**, 28 (2017).

236. Shen, Y., Zhang, H. & Zhao, J. Integrating shared autonomous vehicle in public transportation system: A supply-side simulation of the first-mile service in Singapore. *Transportation Research Part A: Policy and Practice* **113**, 125 (2018).

237. Nourinejad, M., Bahrami, S. & Roorda, M. J. Designing parking facilities for autonomous vehicles. *Transportation Research Part B: Methodological* **109**, 110 (2018).

238. Wen, J., Chen, Y. X., Nassir, N. & Zhao, J. Transit-oriented autonomous vehicle operation with integrated demand-supply interaction. *Transportation Research Part C: Emerging Technologies* **97**, 216 (2018).

239. Fagnant, D. J., Kockelman, K. M. & Bansal, P. Operations of shared autonomous vehicle fleet for Austin, Texas, market. *Transportation Research Record* **2563**, 98 (2015).

240. Bösch, P. M., Ciari, F. & Axhausen, K. W. Autonomous vehicle fleet sizes required to serve different levels of demand. *Transportation Research Record* **2542**, 111 (2016).

241. Harper, C. D., Hendrickson, C. T., Mangones, S. & Samaras, C. Estimating potential increases in travel with autonomous vehicles for the non-driving, elderly and people with travel-restrictive medical conditions. *Transportation Research Part C: Emerging Technologies* **72**, 1 (2016).

242. Marczuk, K. A., Hong, H. S. S., Azevedo, C. M. L., Adnan, M., Pendleton, S. D., Frazzoli, E., *et al. Autonomous mobility on demand in SimMobility: Case study of the central business district in Singapore* in *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)* (2015), 167.

243. Cheng, S.-F. & Nguyen, T. D. *TaxiSim: A multiagent simulation platform for evaluating taxi fleet operations* in *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02* (2011), 14.

244. Spickermann, A., Grienitz, V. & Heiko, A. Heading towards a multimodal city of the future?: Multi-stakeholder scenarios for urban mobility. *Technological Forecasting and Social Change* **89**, 201 (2014).

245. Bösch, P., Ciari, F. & Axhausen, K. The IVT 2015 baseline scenario. *16th Swiss Transport Research Conference* (2016).

246. Maciejewski, M. & Nagel, K. *Towards multi-agent simulation of the dynamic vehicle routing problem in matsim* in *International Conference on Parallel Processing and Applied Mathematics* (2011), 551.

247. Egbelu, P. J. & Tanchoco, J. M. Characterization of automatic guided vehicle dispatching rules. *The International Journal of Production Research* **22**, 359 (1984).

248. Uber. *Hexagonal hierarchical geospatial indexing system* `https://github.com/uber/h3` [Accessed: 19-Nov-2020]. 2019.

249. Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2**, 83 (1955).

250. Shladover, S. E., Su, D. & Lu, X.-Y. Impacts of cooperative adaptive cruise control on freeway traffic flow. *Transportation Research Record* **2324**, 63 (2012).

251. Levin, M. W. & Boyles, S. D. A multiclass cell transmission model for shared human and autonomous vehicle roads. *Transportation Research Part C: Emerging Technologies* **62**, 103 (2016).

252. Harker, P. T. Multiple equilibrium behaviors on networks. *Transportation Science* **22**, 39 (1988).

253. Levin, M. W. & Boyles, S. D. A cell transmission model for dynamic lane reversal with autonomous vehicles. *Transportation Research Part C: Emerging Technologies* **68**, 126 (2016).

254. Zhang, K. & Nie, Y. M. Mitigating the impact of selfish routing: An optimal-ratio control scheme (ORCS) inspired by autonomous driving. *Transportation Research Part C: Emerging Technologies* **87**, 75 (2018).

255. Bowman, J. L. A comparison of population synthesizers used in microsimulation models of activity and travel demand (2004).

256. Deming, W. E. & Stephan, F. F. On the least squares adjustment of a sampled frequency table when the expected marginal totals are known. *Annals of Mathemtical Statistics* **11 (4)**, 427 (1940).

257. Arentze, T. A., Timmermans, H. J. P. & Hofman, F. Population synthesis for microsimulating travel behavior. *Transportation Research Record* **2014 (11)**, 85 (2007).

258. Ye, X., Konduri, K., Pendyala, R. M., Sana, B. & Waddell, P. A methodology to match distributions of both household and person attributes in the generation of synthetic populations. *Paper presented at the the 88th Annual Meeting of the Transportation Research Board, Washington, D.C.* (2009).

259. Auld, J., Mohammadian, A. K. & Wies, K. An efficient methodology for generating synthetic populations with multiple control levels. *Paper presented at the the 89th Annual Meeting of the Transportation Research Board, Washington, D.C.* (2010).

260. Pritchard, D. R. & Miller, E. J. Advances in agent population synthesis and application in an integrated land use and transportation model. *Paper presented at the the 88th Annual Meeting of the Transportation Research Board, Washington, D.C.* (2009).

261. Srinivasan S., L. M. & Yathindra, K. Procedure for forecasting household characteristics for input to travel-demand models. *Final Report, TRC-FDOT-64011-2008, Transportation Research Center, University of Florida* (2008).

262. Müller, K. & Axhausen, K. W. *Population synthesis for microsimulation: state of the art* in *STRC 2010* (2010).

263. Marini, M. *Agent-based assessment of future demographics and impact on infrastructure transition needs* PhD thesis (ETH Zurich, 2020).

264. Damm, D. in *Recent Advances in Travel Demand Analysis* (eds Carpenter, S. & Jones, P.) 3 (Gower Aldershot, England, 1983).

265. Kitamura, R. An evaluation of activity-based travel analysis. *Transportation* **15**, 9 (1988).

266. Algers, S., Daly, A., Kjellman, P. & Widlert, S. *Stockholm model system (SIMS): Application* in *7th World Conference of Transportation Research* (1995), 16.

267. Torsten, H. What about people in Regional Science? *Regional Science Association* **24**, 6 (1970).

268. Habib, K. M. N. A random utility maximization (RUM) based dynamic activity scheduling model: Application in weekend activity scheduling. *Transportation* **38**, 123 (2011).

269. Pendyala, R. M., Kitamura, R. & Reddy, D. P. Application of an activity-based travel-demand model incorporating a rule-based algorithm. *Environment and Planning B: Planning and Design* **25**, 753 (1998).

270. Arentze, T., Hofman, F., van Mourik, H. & Timmermans, H. ALBATROSS: multiagent, rule-based model of activity pattern decisions. *Transportation Research Record* **1706**, 136 (2000).

271. Roorda, M. J., Miller, E. J. & Habib, K. M. Validation of TASHA: A 24-h activity scheduling microsimulation model. *Transportation Research Part A: Policy and Practice* **42**, 360 (2008).

272. Davidson, W., Donnelly, R., Vovsha, P., Freedman, J., Ruegg, S., Hicks, J. & Castiglione Joand Picado, R. Synthesis of first practices and operational research approaches in activity-based travel demand modeling. *Transportation Research Part A: Policy and Practice* **41**, 464 (2007).

273. Hatzopoulou, M. & Miller, E. J. Linking an activity-based travel demand model with traffic emission and dispersion models: Transport's contribution to aipollution in Toronto. *Transportation Research Part D: Transport and Environment* **15**, 315 (2010).

274. Viegas, J. M. & Martínez, L. M. *Generating the universe of urban trips from a mobility survey sample with minimum recourse to behavioural assumptions* in *Proceedings of the 12th World Conference on Transport Research* (2010).

275. Lam, S. K., Pitrou, A. & Seibert, S. *Numba: A LLVM-based Python JIT compiler* in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (2015), 1.

276. Federal Statistical Office. *Adult Literacy and Life Skills Survey* (Bern, Switzerland, 2003).

277. Federal Statistical Office. *Federal Register of Buildings and Dwellings* (Bern, Switzerland, 2015).

278. Federal Statistical Office / Federal Office for Spatial Development. *Mobility and Transport Microcensus (MTMC) 2015* (Bern, Switzerland, 2015).

279. Federal Statistical Office. *Population and Households Statistics* Bern, Switzerland, 2010.

280. Federal Statistical Office. *Generalized boundaries of local and regional authorities* Neuchâtel, Switzerland, 2018.

281. Federal Statistical Office. *Gemeindetypologie und Stadt/Land-Typologie 2012* (Neuchâtel, Switzerland, 2017).

282. Federal Office for Spatial Development. *ÖV-Güteklassen* Bern, Switzerland, 2019.

283. Federal Office of Topography. *Official directory of towns and cities* Wabern, Switzerland, 2019.

284. Federal Roads Office. *Vehicle owner register (Fahrzeughalterregister)* Bern, Switzerland, 2017.

285. Google. *Google COVID-19 Community Mobility Reports* `https://www.google.com/covid19/mobility/` [Accessed: 15-Jul-2022]. 2020.

286. IVT, E. Z. & WWZ, U. B. *MOBIS Covid19 Mobility Report* `https://ivtmobis.ethz.ch/mobis/covid19/reports/mobis_covid19_report_2020-06-04.html` [Accessed: 15-Jul-2022]. 2020.

287. Wilson, N. H., Weissberg, R. W. & Hauser, J. *Advanced dial-a-ride algorithms research project* tech. rep. (1976).

288. Daganzo, C. F. An approximate analytic model of many-to-many demand responsive transportation systems. *Transportation Research* **12**, 325 (1978).

289. Wilson, N. H. & Hendrickson, C. Performance models of flexibly routed transportation services. *Transportation Research Part B: Methodological* **14**, 67 (1980).

290. Aldaihani, M. M., Quadrifoglio, L., Dessouky, M. M. & Hall, R. Network design for a grid hybrid transit service. *Transportation Research Part A: Policy and Practice* **38**, 511 (2004).

291. Li, X. & Quadrifoglio, L. Optimal zone design for feeder transit services. *Transportation Research Record* **2111**, 100 (2009).

292. Sadowsky, N. & Nelson, E. The impact of ride-hailing services on public transportation use: A discontinuity regression analysis (2017).

293. Brown, A. Not all fees are created equal: Equity implications of ride-hail fee structures and revenues submitted to transport policy. *Transport Policy* (2022).

294. Lazo, L. Ripple effect of Metro's troubles: plummeting bus ridership across the region. *The Washington Post* (2016).

295. Fitzsimmons, E. G. Subway Ridership Declines in New York. Is Uber to Blame. *The New York Times* **23** (2017).

296. Mageean, J. & Nelson, J. D. The evaluation of demand responsive transport services in Europe. *Journal of Transport Geography* **11**, 255 (2003).

297. Kim, S., Ulfarsson, G. F. & Hennessy, J. T. Analysis of light rail rider travel behavior: Impacts of individual, built environment, and crime characteristics on transit access. *Transportation Research Part A: Policy and Practice* **41**, 511 (2007).

298. Babar, Y. & Burtch, G. Examining the heterogeneous impact of ride-hailing services on public transit use. *Information Systems Research* **31**, 820 (2020).

299. Tirachini, Alejandro and Gomez-Lobo, Andres. Does ride-hailing increase or decrease vehicle kilometers traveled (VKT)? A simulation approach for Santiago de Chile. *International Journal of Sustainable Transportation* **14**, 187 (2020).

300. Frank, L. D., Andresen, M. A. & Schmid, T. L. Obesity relationships with community design, physical activity, and time spent in cars. *American Journal of Preventive Medicine* **27**, 87 (2004).

301. Malos, S., Lester, G. V. & Virick, M. Uber drivers and employment status in the gig economy: Should corporate social responsibility tip the scales? *Employee Responsibilities and Rights Journal* **30**, 239 (2018).

302. Oh, S., Seshadri, R., Azevedo, C. L., Kumar, N., Basak, K. & Ben-Akiva, M. Assessing the impacts of automated mobility-on-demand through agent-based simulation: A study of Singapore. *Transportation Research Part A: Policy and Practice* **138**, 367 (2020).

303. Basu, R., Araldo, A., Akkinepally, A. P., Nahmias Biran, B. H., Basak, K., Seshadri, R., Deshmukh, N., Kumar, N., Azevedo, C. L. & Ben-Akiva, M. Automated mobility-on-demand vs. mass transit: a multi-modal activity-driven agent-based simulation approach. *Transportation Research Record* **2672**, 608 (2018).

304. Gurumurthy, K. M., Kockelman, K. M. & Zuniga-Garcia, N. First-Mile-Last-Mile Collector-Distributor System Using Shared Autonomous Mobility. *Transportation Research Record* **2674**, 638 (2020).

305. Stiglic, M., Agatz, N., Savelsbergh, M. & Gradisar, M. Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research* **90**, 12 (2018).

306. Huang, Y., Kockelman, K. M. & Garikapati, V. Shared automated vehicle fleet operations for first-mile last-mile transit connections with dynamic pooling. *Computers, Environment and Urban Systems* **92**, 101730 (2022).

307. Lau, S. T. & Susilawati, S. Shared autonomous vehicles implementation for the first and last-mile services. *Transportation Research Interdisciplinary Perspectives* **11**, 100440 (2021).

308. DELFI e. V. *German National Transportation Schedule* `https://www.opendata-oepnv.de/ht/en/organisation/delfi/start` [Accessed: 15-Jul-2022]. 2021.

309. infas and DLR and IVT and infas 360 and Federal Ministry of Transport and Digital Infrastructure. *Mobility in Germany (2017)* (Bonn, Berlin, Germany, 2019).

310. Federal Highway Research Institute. *Automatische Zählstellen auf Autobahnen und Bundesstrassen* `https://www.bast.de/DE/Verkehrstechnik/Fachthemen/v2-verkehrszaehlung/zaehl_node.html` [Accessed: 22-Jun-2022]. Bergisch Gladbach, Germany.

311. Mocanu, T., Joshi, J. & Winkler, C. A data-driven analysis of the potential of public transport for German commuters using accessibility indicators. *European Transport Research Review* **13**, 1 (2021).

312. Sevcıková, H., Raftery, A. E. & Waddell, P. A. Assessing uncertainty in urban simulations using Bayesian melding. *Transportation Research Part B* **41**, 652 (2007).

313. Beykaei, S. A. & Miller, E. Testing uncertainty in ILUTE—an integrated land use-transportation micro-simulation model of demographic updating. *Journal of Civil & Environmental Engineering* **7** (2017).

314. Hülsmann, F., Gerike, R. & Ketzel, M. Modelling traffic and air pollution in an integrated approach–the case of Munich. *Urban Climate* **10**, 732 (2014).

315. Zhang, L., Yang, W., Wang, J. & Rao, Q. Large-scale agent-based transport simulation in Shanghai, China. *Transportation Research Record* **2399**, 34 (2013).

316. Bekhor, S., Dobler, C. & Axhausen, K. Integration of activity-based and agent-based models: case of Tel Aviv, Israel. *Transportation Research Record* **2255**, 38 (2011).

317. Kickhofer, B., Hosse, D., Turnera, K. & Tirachinic, A. *Creating an open MATSim scenario from open data: the case of Santiago de Chile* tech. rep. (Technical report, VSP Working Paper 16-02, 2016).

318. Balakrishna, R., Antoniou, C., Ben-Akiva, M., Koutsopoulos, H. & Wen, Y. Calibration of microscopic traffic simulation models: Methods and application. *Transportation Research Record* **1999**, 198 (2007).

319. Hourdakis, J., Michalopoulos, P. & Kottommannil, J. Practical procedure for calibrating microscopic traffic simulation models. *Transportation Research Record* **1852**, 130 (2003).

320. Hollander, Y. & Liu, R. The principles of calibrating traffic microsimulation models. *Transportation* **35**, 347 (2008).

321. Theil, H. *Economic forecasts and policy / by H. Theil, assisted by J.S. Cramer. H. Moerman and A. Russchen* (North-Holland Publishing Company, Amsterdam, 1958).

322. Brockfeld, E., Kühne, R. & Wagner, P. Calibration and validation of microscopic traffic flow models. *Transportation Research Record* **1876**, 62 (2004).

323. Dowling, R., Skabardonis, A. & Alexiadis, V. *Traffic analysis toolbox volume III: guidelines for applying traffic microsimulation modeling software* tech. rep. FHWA-HRT-04-040 (2004).

324. Great Britain Highways Agency. *Design manual for roads and bridges. Volume 12. Section 2. Traffic appraisal advice. Part 1. Traffic appraisal in urban areas* (HMSO, 1996).

325. Roads and Maritime Services, New South Wales Government. *Traffic modelling guidelines* (2013).

326. NZ Transport Agency. *Transport model development guidelines* (2014).

327. Vora, M. N. *Hadoop-HBase for large-scale data* in *Proceedings of 2011 International Conference on Computer Science and Network Technology* **1** (2011), 601.

328. Shvachko, K., Kuang, H., Radia, S. & Chansler, R. *The Hadoop Distributed File System* in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (2010), 1.

329. Adu-Gyamfi, Y. GPU-enabled visual analytics framework for big transportation datasets. *Journal of Big Data Analytics in Transportation* **1**, 147 (2019).

330. Heer, J. & Shneiderman, B. Interactive dynamics for visual analysis. *Communications of the ACM* **55**, 45 (2012).

331. Lavigne, V. & Gouin, D. Visual analytics for cyber security and intelligence. *The Journal of Defense Modeling and Simulation* **11**, 175 (2014).

332. Hashem, I. A. T., Chang, V., Anuar, N. B., Adewole, K., Yaqoob, I., Gani, A., Ahmed, E. & Chiroma, H. The role of big data in smart city. *International Journal of Information Management* **36**, 748 (2016).

333. Barns, S. Smart cities and urban data platforms: Designing interfaces for smart governance. *City, culture and society* **12**, 5 (2018).

334. Wessels, A., Purvis, M., Jackson, J. & Rahman, S. *Remote Data Visualization through WebSockets* in *2011 Eighth International Conference on Information Technology: New Generations* (2011), 1050.

335. Fishman, G. S. *Discrete-event simulation: modeling, programming, and analysis* (Springer Science and Business Media, 2013).

336. Tokyo Metropolitan Area Transportation Planning Council. *Person Trip Survey (Japan)* 2008.

337. National Statistics Center. *Statistics of Japan* `https://www.e-stat.go.jp/en/` [Accessed: 15-Jul-2022]. 2021.

338. Kawasaki, S. The challenges of transportation/traffic statistics in Japan and directions for the future. *IATSS research* **39**, 1 (2015).

STUDENT PROJECTS SUPERVISED

BACHELOR THESES

2019 *Optimization of fleet deployment – Austrian case*
David Ankenbrand

SEMESTER PROJECTS

2017 *Visualization of large-scale simulation data – Where big data meets big world*
Bühler Pascal

2020 *Austria EV infrastructure assessment with agent-based daily-activity model*
Samuel Derwort

2020 *Transition to e-Mobility in California, US*
Dennis Bader

2021 *Agent-based modelling of passenger demand in airports*
Micha Weber

2022 *Data mining large-scale agent-based energy and mobility simulations*
Tim Brand

2022 *Fusing real-time data into agent-based simulations of energy and mobility transitions*
Philipp Rohrer

MASTER THESES

2020 *Large-scale freight traffic modelling: An agent-based approach*
Patrick Maire

2021 *Improved learning of agents from previous travel experience*
Nuha Lindon

CURRICULUM VITAE

PERSONAL DATA

Name	Aleksandr Saprykin
Date of birth	September 27, 1988
Place of birth	Leningrad, USSR
Citizen of	Russian Federation

EDUCATION

2016 – 2023	ETH Zürich, Laboratory for Energy Conversion, Zürich, Switzerland PhD Candidate
2014 – 2016	ITMO University, Saint Petersburg, Russian Federation PhD Candidate
2005 – 2011	Saint Petersburg State Electrotechnical University "LETI" Saint Petersburg, Russian Federation *Final degree:* Specialist diploma (information security)

EMPLOYMENT

2015 – 2016	Software Engineer *Special Technologies Ltd*, Saint Petersburg, Russian Federation
2010	Internship *Google Summer of Code*, On behalf of GNOME Foundation and GStreamer team
2009 – 2015	Software Engineer *JSC "Academician A.L. Mints Radiotechnical Institute"*, Saint Petersburg, Russian Federation

PUBLICATIONS

Articles in peer-reviewed journals:

1. Saprykin, A., Chokani, N. & Abhari, R. S. GEMSim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios. *Simulation Modelling Practice and Theory* **94**, 199 (2019).
2. Saprykin, A., Chokani, N. & Abhari, R. S. Uncertainties of sub-scaled supply and demand in agent-based mobility simulations with queuing traffic model. *Networks and Spatial Economics* **21**, 261 (2021).
3. Plagowski, P., **Saprykin**, A., Chokani, N. & Shokrollah-Abhari, R. Impact of electric vehicle charging–An agent-based approach. *IET Generation, Transmission & Distribution* **15**, 2605 (2021).
4. Saprykin, A., Chokani, N. & Abhari, R. S. Accelerating agent-based demand-responsive transport simulations with GPUs. *Future Generation Computer Systems* **131**, 43 (2022).
5. Joubert, C. J., **Saprykin**, A., Chokani, N. & Abhari, R. S. Large-scale agent-based modelling of street robbery using graphical processing units and reinforcement learning. *Computers, Environment and Urban Systems* **94**, 101757 (2022).

Conference contributions:

6. Saprykin, A., Chokani, N. & Abhari, R. S. *Large-scale multi-agent mobility simulations on a GPU: towards high performance and scalability* in. **151** (Elsevier, 2019), 733.
7. Saprykin, A., Chokani, N. & Abhari, R. S. *Gridlock resolution in a GPU-accelerated traffic queue model* in. **170** (Elsevier, 2020), 681.
8. Saprykin, A., Marini, M., Chokani, N. & Abhari, R. S. *Holistic, integrated generation of daily-activity plans for Switzerland: from population synthesis to trip generation* in *20th Swiss Transport Research Conference (STRC 2020)(online)* (2020).
9. Saprykin, A., Chokani, N. & Abhari, R. S. *A data-driven approach to run agent-based multi-modal traffic simulations on heterogeneous CPU-GPU hardware* in. **184** (Elsevier, 2021), 720.
10. Saprykin, A., Chokani, N. & Abhari, R. S. *Impacts of downscaled inputs on the predicted performance of taxi fleets in agent-based scenarios including Mobility-as-a-Service* in. **201** (Elsevier, 2022), 574.