ETHzürich

Analysing and exploiting Google's FLoC advertising proposal

Master Thesis

Author(s):
Turati, Florian

Publication date:
2022-02

Permanent link:
https://doi.org/10.3929/ethz-b-000539945

Rights / license:
In Copyright - Non-Commercial Use Permitted

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Analysing and exploiting Google's FLoC advertising proposal

Master Thesis

Florian Turati

February 28, 2022

Advisors: Prof. Dr. David Basin, Dr. Carlos Cotrini, Karel Kubicek

Department of Computer Science, ETH Zürich

Abstract

Google proposed to use Federated Learning of Cohorts (FLoC) as a replacement of third-party cookies allowing interest-based advertising while preventing identification of individual users. It divides the users into cohorts according to their browsing history patterns using a clustering algorithm SimHash. However, the proposal received criticisms for not meeting the claimed privacy protections.

Based on our analysis of FLoC, we implement two attacks breaking FLoC's anonymity properties. The first is a Sybil attack by finding preimages of the non-cryptographic hash function SimHash, breaking FLoC's k-anonymity. The second and more serious attack also leverages SimHash, but it can in general work for any clustering algorithm. This attack extracts part of the browsing history of FLoC users just from the generic distribution of browsing patterns and the SimHash. We do so by generating a plausible browsing history for a target hash using a Generative adversarial network (GAN) and the preimage attack. We then evaluate the feasibility of the attack as well as the quality of the generated histories with various metrics. The insights from this attack should help improve the design of FLoC and future proposals.

Acknowledgements

Contents

Chapter 1

Introduction

1.1 Motivation

Websites can count on advertising to offer content and gain revenue. To make this source of income more profitable, the ads shown are usually chosen to be of interest to the user. In order to display relevant ads to users, it is standard to collect data about them, to infer any meaningful information from what they browse online. Common tracking tools include third-party cookies, device fingerprinting etc.

These tracking mechanisms allow third-parties to record user behaviors online and profile them. This causes serious privacy threats. Consequently, partial mitigations have already been put in place. For example, present-day browsers allow blocking third-party cookies as a privacy setting, if not by default [66].

Furthermore, governments address ubiquitous data collection by privacy regulations. For example, country regulators in the European Union are moving against Google Analytics and Facebook Connect [11], citing breach of GDPR and other regulations that would make their usage illegal. It is unlikely that websites are gonna suddenly stop relying on Google Analytics, but more legal issues are arising and companies are fined. Recently, France fined Google and Facebook over 200 million euros for making it too disorienting to refuse the use of cookies [6]. France, following an analysis with other European countries, also made the use of Google Analytics illegal [15] without further regulations.

Studies [9] have shown that removing cookies could cost websites a big part of their revenue (between 50 to 70%). However, it is questionable what part of this revenue growth returns to publishers [63]. Google knows that third-party cookies are decaying and it wants to find a substitute that would still generate considerable revenue while maintaining an acceptable level of

privacy protection. To allow websites to continue offering content for "free" to visitors, Google plans to keep personalized advertisement alive.

As a complement to Google's planned deprecation of third-party cookies, they propose a potentially more private form of tracking named Federated Learning of Cohorts (FLoC). It makes use of an unsupervised clustering algorithm, which divides users into groups (cohorts) of cardinality at least k to achieve k-anonymity. Each user is then assigned an identifier shared with the whole cohort, this identifier is given to websites that implement the FLoC API to perform cohort tracking and targeted advertisement without the need to track singular users. FLoC uses browsing history data to compute a locality sensitive hash (SimHash) in the browser and this hash is then used by Google's trusted server for assignment of the cohort.

However, Google's proposal raised a lot of concerns from competitors, privacy and ad-revenue oriented media. Privacy enthusiasts were quick to point out issues and possible attacks on FLoC [17, 52]. For example, browsing histories tend to be stable over time and can be used as fingerprint to uniquely identify users [46]. Consequently, competing browsers disabled FLoC [56, 59] and some websites opted-out of FLoC (default opt-in) trial [1]. FLoC became controversial both for users and advertisers. The latter questioned Google's claim that FLoC is 95% as effective as cookies [54].

1.2   Contributions

In this work, we propose and demonstrate two attacks on this new proposal. The first is a preimage attack on the non-cryptographic SimHash. This allows us to generate random histories that match a target SimHash, and therefore we can break k-anonymity of a target cohort using the Sybil attack [21].

Another attack focuses on reverse-engineering the browsing history from a user's SimHash. Generative Adversarial Networks (GANs) are already known to be capable of generating realistic faces that are indistinguishable from real faces and judged more trustworthy [44]. We can thus make use of GAN techniques to generate plausible browsing histories. Since browsing histories are rather stable over time we can use an existing dataset to train it. However, for privacy reasons we do not use a dataset of browsing histories, but rather a dataset of users' watched movie list that was also used by Google to evaluate FLoC.

We then further narrowed the history generated by the GAN using the preimage attack such that it matches the target SimHash. We propose metrics to evaluate the GAN's movie history generation. Those techniques can be adapted to a GAN generating browsing history of other kinds of similar user data. We take random history generation as a baseline. Such metrics include:

- Comparing the number of common movies between the generated and real movie history.

- Computing the minimum Hamming distance between the generated histories and every history in the test set.

- We can also use one-dimensional Wasserstein distance. It compares the distribution of appearance counts for each movie, between the test data and a similar number of generated histories.

- And finally, we reproduce the metric used by Google to evaluate their FLoC proposal [50]. This metric measures the "utility of cohorts for varying levels of anonymity". It uses real users and compares the utility of cohorts, meaning it tries to compare how similar users are in each cohort. We also compute this metric with users generated by our GAN and compare our generated users with real users in the same cohort.

For each metric, we show how successful our history reconstruction attack is.

1.3 Report Structure

The remainder of this work is arranged as follows: Chapter 2 explains the FLoC proposal, its objectives, implementation, shortcomings and how we model it. Chapter 3 describes the implementation of the components of our attack and how it breaks k-anonymity. Chapter 4 presents the results of the evaluation of the components of our attack. Chapter 5 reviews the related work in the domain of privacy-preserving interest based advertisement and similar attacks. Lastly, Chapter 6 concludes this work.

Chapter 2

FLoC Proposal

2.1 Goal

2.1.1 Utility

The README at FLoC's GitHub repository [64] defines three categories of advertisement:

1. First-party and contextual information (e.g., "put this ad on web pages about motorcycles").

2. General information about the interests of the person who is going to see the ad (e.g., "show this ad to Classical Music Lovers").

3. Specific previous actions the person has taken (e.g., "offer a discount on shoes that the user left in a shopping cart").

Federated Learning of Cohorts (FLoC) fits in the second category. It is supposed to replace tracking third-party cookies (used for the second and third advertising categories) while still allowing ads to be related to users' interests.

A first-party cookie's domain attribute matches the domain of the currently browsed website. It is not the case for third-party cookies. They enable tracking of a browser's history to the benefits of advertisers to display pertinent ads. Bashir et al. [5] showed that advertisers are able to see at least 91% of average user's browsing history.

FLoC is a proposal to address the ads shown based on a user inferred overall interest, partly replacing third-party cookies. Other proposals in the privacy sandbox [38] help phase out third-party cookies completely.

2.1.2 Privacy

The FLoC proposal aimed to achieve k-anonymity, despite the fact that differential privacy is becoming the standard privacy notion in industry and academia. Google researchers provided more information on this choice in their whitepaper [50] as "[they] believe it fails to quantify the hardness of tracking users across the web".

In the context of FLoC, k-anonymity of cohorts means that the FLoC ID is shared by at least k users. And that a user can hide among this set of at least $k - 1$ other users to avoid being uniquely identified. It is known that homogeneity attacks can break k-anonymity. If all users in a cohort share some information (e.g., visiting some website about an unlikely medical condition), then we also learn that information. As a countermeasure, Google proposed to measure the sensitivity of cohorts with the privacy notion of t-closeness [2].

The main goal of Google with its FLoC proposal is to find a replacement for third-party cookies that will still allow advertisers to generate revenue while being less privacy invasive. The parameters like the cohort size and the t-closeness are part of the privacy-utility trade-off assessed by the Google team.

2.2 FLoC Implementation

2.2.1 FLoC ID Computation

In this section we give an overview of the Chromium FLoC ID computation. More details can be found in [49].

Encoding the browsing history   For a given user, Chromium computes the FLoC ID of the cohort by taking as input the user's browsing history. Websites in the browsing history for the past seven days are encoded by taking the hash of the domain names. The only exception are websites that are marked as sensitive and these are therefore filtered from the computation.

Computing the SimHash of the encoding   SimHash is applied on the encoded domain name set. SimHash is an algorithm from the locality-sensitive hashing (LSH) family [50]. SimHash maps similar vectors to the same hashes with higher probability than dissimilar vectors. Therefore, related browsing history are more likely to map to the same hash. The output SimHash length was set to 50 bits of which only a prefix is used.

Ensuring k-anonymity   The computation until now can be done in browser without sharing the history. However, it is not possible to ensure k-anonymity

without knowing how many users are in each cohort. A central server (owned by Google in the current proposal) is needed to distribute the mapping of SimHashes to FLoC cohort IDs. This server ensures each cohort has at least the required minimum size and can remove sensitive cohorts based on the t-closeness parameter. Depending on the minimum cohort size, all bits of the SimHash might not be needed. Therefore, users with different SimHash with a common prefix may be mapped to the same cohort.

From this overview it can be noted that despite its name, **FL**oC does not include **F**ederated **L**earning. More details on some of these steps will be provided when needed.

2.2.2 Origin Trial

In 2021 from April to July, Google evaluated the initial version of FLoC in an origin trial. In selected countries 0.5% random users of Chrome version 89 to 91 were selected for the preliminary study.

A website can access the FLoC ID via a browser API. It receives an integer, which serves as a unique identifier for a cohort. A cohort contains users sharing similar browsing histories and by extrapolation interests. FLoC does not provide explicit information on a users' history. However, different websites can share the history they observed of users with the same FLoC ID. This allows them to infer the union of histories from users in a cohort.

The origin trial for FLoC was conducted with the algorithm explained previously in Section 2.2.1. In [14], some of the parameters used are presented. The cohort were calculated once every seven days and included seven days of browsing history (at least seven different sites are required for computation). The websites that were included in the FLoC computations either displayed ads or called the FLoC API. The FLoC cohort identifiers were global across websites. It was also possible to opt out but every one (websites and users) qualifying for the origin trial was included by default.

Google reported limited insights from data collected during the origin trial [49]: Before any filtering was applied the number of cohorts was 33 872. Between 13 and 20 bits from SimHash were needed to define a cohort. The minimum number of users in a cohort was 2 000. Among those users in a cohort there are at least 735 different sets of visited domains. Each user has one set of visited domains and sometimes several users have the same set. The t-closeness test ($t = 0.1$) filtered 792 sensitive cohorts (approximately 2.3%).

Advertisers also took advantage of the origin trial to analyze the data they collected and potentially give suggestions on how to improve FLoC. A few reported some of their findings. CafeMedia [40] is an advertiser which works with more than 3000 websites. Not enough data was available to perform

cafe
media

10 Content Keywords That Over-Index Against Each kFLoC

0	music	support	grade	questions	season	travel	team	pictures	cream	deals
1000	dogs	guides	working	things	roast	state	shrimp	people	articles	girl
2000	writing	magic	vegetables	movies	slow	japanese	cooker	history	oatmeal	beans
3000	prime	high	rolls	magic	chili	animals	build	shows	dinner	meal
4000	weekly	world	disney	magic	sheets	pancakes	photos	more	modern	update
5000	kitchen	tips	word	meal	support	crochet	parmesan	star	things	animals
6000	paper	cheesecake	help	good	wedding	girl	prices	strawberry	cook	should
7000	support	vegetable	windows	favorite	photos	watch	types	release	gluten	plant
8000	craft	paper	card	deals	growing	instant	blueberry	school	return	broccoli
9000	stuffed	ingredient	breakfast	dogs	more	growing	projects	peanut	corn	sugar
10000	classic	chip	coffee	honey	coconut	weekly	vegan	card	product	dressing
11000	bars	list	player	names	italian	dinner	beef	chip	cheesecake	articles
12000	model	activities	favorite	kitchen	avocado	wedding	space	science	drive	american
13000	shows	fried	girl	prices	keto	wedding	search	printable	ingredient	support
14000	pictures	model	movies	spinach	stuffed	school	business	word	working	honey
15000	codes	printable	dressing	season	state	eggs	spinach	york	card	questions
16000	table	business	weekly	deals	drive	iphone	mexican	season	york	room
17000	state	smoothie	school	cooker	park	slow	chili	beans	eggs	vegetables
18000	prime	wedding	build	meal	music	good	york	business	parmesan	animals
19000	quick	mexican	drive	banana	sausage	guides	avocado	creamy	season	breakfast
20000	subscriber	crochet	pattern	games	writing	summer	gear	release	fryer	science
21000	pancakes	change	travel	noodles	gear	school	more	review	watch	casserole
22000	build	category	classic	dinner	online	county	apps	minute	shows	weekly
23000	japanese	cars	down	prices	girl	photos	need	cinnamon	using	modern
24000	word	dishes	animals	projects	activities	paint	park	quick	writing	android
25000	reveal	people	life	excel	tacos	tomato	calculator	wedding	deals	prices
26000	dogs	perfect	plant	broccoli	weekly	first	grow	cook	sausage	return
27000	player	team	magic	salad	broccoli	greek	potato	projects	american	apps
28000	product	pattern	crochet	spring	high	word	subscriber	search	school	apple
29000	modern	gear	work	model	cars	japanese	star	coffee	android	search
30000	dishes	thai	park	state	cinnamon	strawberry	vegetable	install	peanut	smoothie
31000	paint	movies	county	smoothie	banana	muffins	favorite	room	dressing	change
32000	healthy	oatmeal	love	tomato	casserole	beans	apps	travel	calculator	copycat
33000	table	games	minute	gear	update	excel	print	thai	business	greek

Figure 2.1: CafeMedia keyword classification of cohorts

statistical analysis on every recorded FLoC ID (around 34 000). But using the property of SimHash (similar vectors are more likely to map to the same value), and the fact that ranges of SimHashes are mapped to cohort ids, FLoC IDs can be grouped together. In their analysis they split contiguous cohort ids into groups of 1 000. With that they were able to extract meaning from those groups of cohorts. A result of their analysis is shown in Fig. 2.1. It summarizes the content browsed by members of cohorts in 10 keywords. It shows however that the extracted insights during FLoC trial are quite limited.

Criteo [53] is another advertiser which posted the insights it gained during the origin trial. They accumulated data from more than 16 000 advertiser websites and 2 000 publisher websites over one and a half month. They noted that not a lot of data was available, only 0.02% (Google announced 0.5%) of Chrome users participated in the origin trial since it runs on the Beta and Dev versions. Only 10 countries were part of the origin trial (it might not be compliant with some countries regulation, e.g., GDPR). This makes it difficult to draw meaningful information from the analysis. They logged on average, over a seven day period, 2.8 unique users per cohort. The composition of cohorts changes a lot each seven days, only 12% of users were still in the same cohort after a change. And the average variation of FLoC ID when users

change cohort is 9600, the range of cohort identifiers being $[0, 34\,000]$. From their perspective FLoC may not be seen as effective as advertised by Google. However, with the lack of transparency from Google it is hard to verify their claim. This may again be due to the small number of users participating in the origin trial.

The origin trial of FLoC stopped after Chrome version 91. Google stated they were working on improving FLoC based on the feedback received before further testing.

2.3 Threat Model

By design FLoC leaks information about the users. FLoC's primary goal is to track users and it works in accordance with a privacy-utility trade-off. Google advertises that the deprecation of third party cookies along with the arrival of FLoC (and other privacy sandbox proposals [38]) will provide more privacy to end users than the status quo. In Chapter 5, we will see if this claim can be validated. From the various analyses of the FLoC proposal we found, only one seemed to explicitly define a threat model [52].

When defining our threat model we should think of strong adversaries that may arise. Given that Google is an advertiser, tracker, Chrome vendor, and the company behind FLoC, it is reasonable to consider Google as the strongest adversary with potential to access more data than any other party. With FLoC, Chrome is developing the tool for targeted advertisements, but Google also displays relevant ads on their own websites. There is a conflict of interests and Google can be both the defender and the attacker. Chrome is responsible for the assignment of SimHashes to cohort, therefore it can see all the SimHashes which others (e.g., websites receiving a FLoC ID) would not. Google also decides on the mapping from SimHash to cohort ID, for example which cohorts to remove due to t-closeness. Other parties would only receive a FLoC ID and not know the user's SimHash. Since a minimum size is required for the cohorts, multiple SimHashes can be mapped to the same cohort. It is also important to remember Google's dominant position. Chrome is by far the most used browser [58]. They utilize this position by, for example, acquiring user information via the Chrome User Experience Report.

The second considered adversary has less information to work with. It would be a malicious party, for example websites that use the FLoC API to receive visitors' FLoC IDs. They can only approximate the SimHash originating from the FLoC ID they observe. However, that information is sufficient to target groups or even individuals.

In summary, we have two adversaries:

- *Strong Adversary:* Google with a possibility to observe every SimHash of users' histories.

- *Weak Adversary:* It has a partial view and wants to approximate the SimHashes.

2.4 Possible FLoC Improvements

Since the FLoC proposal, a lot of criticism (discussed in Chapter 5) has arisen and some of it included suggestions for possible improvements. During a talk at an IETF conference in July 2021, Josk Karlin [14] summarized some of the improvements they planned for the next iteration of FLoC. We summarize some of those improvements below.

Considering the feedback the next version should not be opt-in by default for websites with ads. During the origin trial the qualifying users that were included did not specifically opt-in. They did however have to allow sharing information prior with Google to be qualified.

Another issue was related to the meaning of cohorts. If a user or an advertiser sees the FLoC ID 13729, a priori he would not have much understanding about what that corresponds to. To make cohort identifiers more meaningful they consider providing topics (extracted from visited domains) instead. Topics can be selected, which can help with the server-side t-closeness analysis to remove the one correlated with sensitive information. Users can then better comprehend what FLoC is sharing about them and potentially opt out of particular topics.

The FLoC ID during the origin trial provided approximately 16 bits of entropy for browser fingerprinting. Using selected topics instead of cohorts they could reduce it to around 8 bits, which is still a significant amount of information. Therefore they thought about including some differential privacy mechanisms. For example returning a random FLoC ID with small probability or giving different IDs to different webpages.

The possible improvements the Chrome team was considering were based on some of the feedback they received (e.g., Github issues [64], Mozilla report [52]). Finally in early 2022 FLoC was replaced by the Topics API [30], which factored in some of the improvements. As of now the proposal has been presented and is open to feedback. However, there is still not much detail on its implementation. Since Topics API was released just briefly before finishing this thesis, we do not inspect it and focus only on FLoC.

Chapter 3

Breaking k-anonymity

This chapter's sections follow the chronological order in which the different approaches were tried. We first find preimages (e.g., domain names) for a given target SimHash. This allows us to generate random histories matching a target SimHash. It is then possible to mount a Sybil attack and isolate real users in the cohort, breaking *k*-anonymity. In a second attack, we generate plausible user histories for a target hash using a GAN and the preimage attack. This attack also leverages SimHash, but can in general work for any clustering algorithm.

3.1 FLoC basic functionality re-implemented

In Section 2.2.1 we gave an overview of the steps needed to compute a FLoC ID. For the purpose of our attack and to gain more insight into the inner working of Chromium's version of FLoC, we reimplemented them in Python. In this section we give more details to some parts that we will need later.

We use the Chromium source code as reference[1] and also a FLoC simulator in Go [45], which helped abstract away the components that were not necessary for FLoC. We debugged Chromium browser in a version satisfying the origin trial requirements to test the correctness of the computations.

3.1.1 CityHash

CityHash [48] is a hash function used for processing the input domain names to pseudorandom numbers needed during the SimHash computation. Note that CityHash has a known vulnerability [19]. We do not utilize this vulnerability, but if we would, it could make our attack more powerful.

---

[1] https://github.com/chromium/chromium/tree/d7da0240cae77824d1eda25745c4022757499131/components/federated_learning

For this particular legacy hash function we took a Python implementation [55] and corrected some minor errors. An interesting error happens when computing CityHash for movie names with non-latin alphabet, where it returns hashes longer than the expected 64 bit length. To solve this problem we just truncated the hash to 64 bits.

3.1.2 SimHash

In Chromium, SimHash takes as input a set of strings (e.g., domain names) and for each of them it computes a fingerprint vector of the desired output length. This fingerprint vector for a domain in FLoC is 50 randomly sampled values from a Gaussian distribution with the domain's hash used as a seed for the sampling. The number 50 is the output length parameter set by FLoC authors. As the output it computes the sum of all these vectors and applies the sign function (1 if input is positive, 0 otherwise) to get the resulting binary vector.

Going more into the pseudo-random Gaussian generator, it uses the City-Hash of the current domain and the bit position as seed. For our attack in Section 3.2, we will use the Gaussian generator with a call to the function `random_gaussian(bit_pos, domain_hash)`. This function applies a Box-Muller transform to sample from a Gaussian using two uniformly distributed random numbers.

3.1.3 Prefix LSH

Prefix LSH [49, 23] is a variant of the proposed Sorting LSH in the FLoC whitepaper [50]. Since the amount of users sharing the same SimHash may be insufficient, multiple contiguous SimHashes can map to the same cohort, in a cohort the SimHash bitstrings share the same prefix. Prefix LSH defines the ranges of similar SimHashes that will be mapped to the same FLoC ID. This operation is performed server-side as it needs to keep track of the cohort size to ensure *k*-anonymity and filter out the blocked cohorts. The cohort assignment is then periodically sent to the browser.

Taking the same analogy as [52], we can think of Prefix LSH as building a binary tree. Starting with the most significant bit, we get two branches and we grow the tree as long as each subtree contain the required minimum number of users.

For our attack we decided to compute preimages directly on SimHash and not the resulting FLoC ID as the latter seemed to be more prone to change than the SimHash computation. Since a Chrome server distributes a file to the browser that defines the mapping from SimHash to FLoC ID. At the same time it abstracts away the Prefix LSH algorithm which is then performed on the Chrome-operated central server. Although Google stated

they will experiment with different clustering algorithms, none others were tried publicly.

3.2 Preimage Attack on SimHash

3.2.1 Greedy Heuristic

First, we begin with a simple idea to break the non-cryptographic hash function used in Chromium's source code for the FLoC computation. SimHash (see Section 3.1.2) uses a pseudo-random Gaussian generator with a domain hash as seed. This observation gives rise to the following heuristic: For each output bit, draw a domain name from a set. Using the domain's hash as a seed for the Gaussian generator, repeatedly sample from a Gaussian until an outlier is found. The sign of the outlier Gaussian sample depends on the target SimHash bit (positive if the current target is 1, otherwise negative). With a high probability, the large value will be enough to flip the sign of the sum to the target value, since the mean of the normal distribution is 0, so other random inputs preserve this large value.

Note that we use two different sampling processes. The first one *draws* uniformly at random a domain name from a set and is not part of the SimHash API. The other one uses the SimHash API to *sample* from a Gaussian using the previously sampled domain as a seed.

We propose a greedy heuristic [16], since sampling an outlier from a Gaussian for the bit under consideration is a greedy choice. Nonetheless, we do not have guarantees that we can find an acceptable solution as a priori our problem does not show an optimal substructure. In addition, we might have to reconsider our choices, as it can happen that the large (absolute) value we sampled gets overthrown by the other values and the final sum flips sign and flips the desired output bit. To address that problem, having no guarantees on the optimality of the choice we make, we restart the program after a fixed number of attempts to sample a large Gaussian value.

The probability of finding a solution decreases with the increasing number of bits of the hash. Conveniently, the preimage attack still performs well for the bit range of SimHash used during the FLoC origin trial (13-20) [49] and performs decently even for a greater bit count. Also, the SimHash output bit length cannot exceed 64 bits with the current Chromium implementation. There are 3.2 billion Chrome users according to [58] and FLoC announced minimal cluster size of 2000 users during origin trial [49]). Therefore, the SimHash length of more than 21 bits seems unnecessary as:

$$2^{21} \times 2000 = 4\,194\,304\,000 \tag{3.1}$$

Besides, with 21 bits we can easily accommodate all current Chrome users and with 22 bits the current world population (roughly 8 billions people [67]).

3.2.2 Resulting Algorithm

In more detail the algorithm works as follows (we also refer to the step in the python pseudocode in Listing 1):

We have as input the target SimHash with the target prefix bit length we want to match.

We keep track of the current Gaussian sums (one per bit under consideration) and the domain hashes we chose (step 1).

For each bit, we first check if the current target bit matches the sign of the current sum of Gaussian samples (step 2 with `match_target_bit` function). If the bits match using the current domain list, we go to the next bit. Otherwise, we have to add another domain hash that ensures a match of the target bit. In the special case of the first iteration, the set of current hashes is empty, so we consider a mismatch of the first bit and proceed to the step 3.

Since the prefix bits starting from the current one do not match the target bits, we update the threshold for sampling an outlier Gaussian. To update this threshold (step 3 with `update_thresh` function), we use the following formula assuming a defined minimum threshold parameter and access to the current sum for the current bit position:

$$\text{thresh} = \max\left\{\text{min\_threshold}, |\text{gaussian\_sums}\,[\text{bit\_pos}]|\right\}$$

After updating the threshold, we call the subroutine to find an outlier Gaussian sample (step 4 with `find_outlier_gaussian_sample` function). This will be discussed further in Section 3.2.3.

Then we check that with the new sampled value the current sign of the Gaussian sum matches the target bit for the current position (step 5). If it is the case, we check that all the previous bits still match the intended target bit with the updated Gaussian sums (step 6 with `match_previous_target_bits` function). If it is so, we update the Gaussian sums and domain hashes accordingly (step 7 with `update` function), otherwise we repeat the sampling (step 4).

The algorithm terminates when it obtained a list of domain hashes that match the target SimHash. With this preimage attack we can generate random histories matching a target SimHash to perform a Sybil attack, breaking FLoC's *k*-anonymity.

3.2.3 Extensions

There is potential for improvement to this simple attack. For example, we can use more than one outlier with the correct sign for the current bit to increase the probability that the sign is preserved despite all subsequent changes.

Listing 1 Preimage attack on SimHash

```python
def find_preimage(target_simhash, prefix_bitlen):
  # step 1
  gaussian_sums = [0] * prefix_bitlen
  domain_hashes = []
  for bit_pos in range(prefix_bitlen):
    target_bit = target_simhash[bit_pos]
    # step 2
    prefix_bits_mismatch = match_target_bit(bit_pos, target_bit,
    ↪  domain_hashes, gaussian_sums)

    while prefix_bits_mismatch:
      # step 3
      thresh = update_thresh(bit_pos, gaussian_sums)
      # step 4
      sample = find_outlier_gaussian_sample(bit_pos, target_bit, thresh)

      if not match_target_bit(bit_pos, target_bit, domain_hashes,
      ↪  gaussian_sums):
        # step 5
        continue
      # step 6
      prefix_bits_mismatch = match_previous_target_bits(bit_pos,
      ↪  target_bit, domain_hashes, gaussian_sums)
      if not prefix_bits_mismatch:
        # step 7
        update(gaussian_sums, domain_hashes)

  return domain_hashes
```

Note that we do not try to satisfy some minimum requirement on the number of domain names to be eligible for the FLoC ID computation (that minimum was at seven during the origin trial [49]) but the algorithm could be modified to ensure this by adding domains with low absolute values that likely do not change the Gaussian sum sign.

We did not implement those changes as we later focus more on another method to reverse SimHash. Indeed, we can also reformulate the SimHash problem as an integer program and use a known solver, this will be discussed further in Section 3.5.

Sampling Gaussian Outliers

We now explain in detail the way we sample outliers from a Gaussian. As discussed in Section 3.1.2, we have access to a function `random_gaussian(bit_pos, domain_hash)` which deterministically returns a Gaussian sample using two uniform random variables in the $[0, 1]$ range derived from `bit_pos` and `domain_hash`. We already know the target bit position and cannot change it, so our degree of freedom resides in the chosen domain hash.

A straightforward implementation is to randomly sample values in the output range of the 64-bit CityHash function used to map the domain name string to the domain hash integer. While it works well, it would require us to be able to reverse the CityHash. Indeed, if we want to output a preimage, simply outputting CityHashes of unknown domain names is not that useful to us.

Although Google's CityHash [48] has a well-known security vulnerability [19] that could be exploited and can only make our attack stronger, we wanted a simpler approach that does not rely on the property of the hash function.

So we decided to reduce the number of domains we can sample from, by taking strings from an accessible dataset. We sampled domain names from the Tranco top 1 million domain list (see Appendix B.1 for more detail on the dataset). Then we precomputed the CityHash of domains in the list, to draw hashes for finding outlier Gaussian samples. The attack still works with this domain restriction, although it limits the possible samples in step 4 to 1 million.

Later, as we wanted to use GAN techniques to ensure the preimages to SimHash resemble real user data, we relied on another dataset. For privacy reasons, we did not find a suitable public dataset with user browsing histories. Therefore, we followed Google's FLoC Whitepaper [50] and used the MovieLens 25m dataset (see Appendix B.2) with a history of user watched movies and ratings. While the MovieLens dataset contains enough movies to process it the same way as URLs in Tranco list, the next section defines other processing constraints. These constraints limit the number of movies in MovieLens to 5000, which is not enough samples for the step 4 in Listing 1 Hence, we proposed another preimage attack based on integer programming, which we define later.

3.3 Using GANs to Generate Plausible User Data

Generative Adversarial Networks (*GANs*) [25] are well known for being able to generate new samples from the same distribution as the training data. A GAN consists of two neural networks, a generative network *G* and a discriminative network *D*, which compete against each other in a zero-sum game. The generator *G* learns to produce realistic samples with the objective

to deceive the discriminator *D*, which learns to differentiate between the fake samples and the real data. GANs have been used quite extensively when it comes to image data. They are able to generate photographs equivalent to real ones to the human eye [10, 32]. Nevertheless, when it comes to generating textual data, it becomes more challenging as we switch from continuous data (images) to discrete data (text), while algorithms for training GANs were designed for continuous data.

3.3.1 Choice of GAN

GANs for text generation can handle the non-differentiability of discrete data in different manners that can be grouped into two kinds: those that incorporate reinforcement learning (*RL*) techniques and those that transform the discrete data into continous data.

The first GAN for text generation we used was LeakGAN [27] as it had the best score on the COCO caption dataset (see Appendix B.3) out of other competitors [69, 37]. LeakGAN belongs to the category of GANs using RL techniques for the discrete generation. The use of a policy gradient algorithm (e.g., REINFORCE [65]) enables backpropagation through discrete data and training of the generator, but this induces high variance and mode collapse issues. Mode collapse happens when the GAN lacks diversity in its generated outputs. The generator cannot produce several modes of the input data. For example, if we take the MNIST handwritten digit dataset [18]: A GAN experiencing mode collapse would only be able to generate the digit 6. Failing on every other mode (digit).

LeakGAN learns from usually binary feedback obtained from the discriminator. The discriminator can leak the features it extracted to the generator via a manager module. LeakGAN also aims to improve the sparsity of the guiding signal during training by using Hierarchical RL techniques. It allows for information (e.g., reward signal) to be learnt without having to wait for the generation of the whole sentence. This is especially useful for long sentence generation.

Most of the text-generation GANs still use RNN architectures (e.g., incorporating an LSTM network for LeakGAN). Nevertheless, with the prevalence of Transformer architecture for NLP tasks [61], it can make sense to also test those building blocks in a GAN for text generation. Specifically, if it improves our task of generating a set of movies. Vaswani et al. [61] showed that, for translation tasks, Transformer outperforms convolutional or recurrent networks while also being faster to train due to parallelization.

TILGAN [20] is the second model for text generation. It uses a Transformer-based autoencoder and a GAN in the autoencoder's latent space, hence belonging to the latent feature matching class of text generation GANs. It

reported better results than LeakGAN and other models published in the meantime, referring to FM-GAN [12] as the previous state of the art. Unfortunately, when running the provided code for TILGAN with our preprocessed MovieLens dataset, we did not obtain good generation for the time it trained. We also observed instability in TILGAN's results when reading generated output on the COCO Dataset. Sometimes it produces reasonable sentences, but sometimes it outputs gibberish. For example, it generates sentences with broken syntax and is stuck with repeating words. We observed more stability when we trained with LeakGAN, even though the latter's generated sentences can be less diverse and of worse quality (as outlined in TILGAN paper's Table 2 experimental results). As our main goal is to have a working realistic movie history generation, we prefer to go more in depth on one particular model than in breadth with different models. Therefore, we decided to use LeakGAN for our attack.

Note that in our case, we do not need a model with as many restrictions as needed for generic text generation. Sentences are ordered tuples, so swapping words can change the meaning or break the syntax. However, FLoC operates on unordered sets of browsing histories, so our generation task is not sensitive to order of generated websites (or movies in the example training data). Nevertheless, the additional conditions for the GAN to generate sentences still yields realistic browsing histories for our purposes if we remove duplicates.

As a result, the GAN model we use for our attack might not be the best suited for the task at hand, nonetheless it can still serve as a concrete proof of concept. In addition, with machine learning models getting better and better over time our attack would only become stronger. As a prime example, there are already better GAN models [20, 12] available than the LeakGAN we selected.

3.3.2 Implementation

We used the GAN mostly as a black box without modifying its inner workings. We aimed for a proof that the attack works first, keeping the improvement of the GAN model as future work. Working with LeakGAN as a black box was also caused by computational resources, since training requires time and changing inner modules would involve parameter fine tuning. This would consume more time and computational power for not necessarily better results. Hence, we did not modify the default hyperparameters, except for those necessary to accommodate the change of input dataset. Considering the model we chose was also evaluated by its authors on different datasets, those parameters are meant to be changed.

We use a Text GAN to generate browsing history. However, as discussed in Section 3.2.3, we do not have access to browsing history data for privacy

reasons. Therefore, we decided to use movie ratings data from MovieLens (see Appendix B.2). As this kind of data is slightly different from text, we have to perform some preprocessing to match the data used to train the text GANs we found.

Preprocessing

Encoding movie histories as vectors   The GAN requires numerical values as input. The input text data thus needs to be transformed into numerical values. LeakGAN is parametrized by the maximum sentence length and the number of words the model knows (vocabulary size). Therefore, those two parameters must be small as the model size depends on them. We fixed these parameters, so we can give one number to each word in the vocabulary and a sentence becomes a vector of numbers. The sentences that exceed the maximum length are removed as well as those that contains word not in the vocabulary. Another possibility is to reserve a special tag in the vocabulary for missing words and another tag for padding a sentence to the maximum length.

Defining vocabulary and maximum length   For our purposes we define the movies to be words and the sentences to be histories (e.g., list of watched movies). So the vocabulary size is the number of movies from the whole database that we keep. To choose the movies to keep we can take the most frequent ones in the dataset or use other metrics.

The LeakGAN paper [27] reports for the COCO Image Captions dataset (see Appendix B.3) that they "removed the words with frequency lower than 10 as well as the sentences containing them" to reach a vocabulary of 4 980 words. Then, they build a training set of 80 000 sentences and a test set of 5 000. Our first approach tried to match the train-test split. The maximum sequence length was chosen accordingly by taking only the shortest movie lists in the dataset. Doing so allowed us to take full movie history and not sample a subset. With this approach the maximum sentence length we reached was 77 to get the target 85 000 histories.

This restriction left us with a vocabulary size of around 20 000. To reduce the vocabulary size we removed the movies with less than 20 occurrences to reach a vocabulary size of 5 846. That value is close to the 5 742 that LeakGAN used for Long Text Generation with another dataset. The average sentence length of the other dataset is around 28 while with our approach the average history length is around 41. Hence we can expect to train on longer sequences.

Attempt at giving meaning to encoding   Finally, we also tried to give some meaning to the encoded movie numbers instead of ordering them randomly.

The idea being that the model could learn this ordering and maybe it could improve its generation. One idea is to sort the movies according to genres. The genres are encoded as a bit vector represented by an int, the bit 1 means the genre is present and 0 its absence. If the genres are equal, then the release date is used for sorting, when the release year is not available a default high value is used. Similar processing would be possible with browsing histories, where the websites can be assigned to genres (topics).

Avoiding learning order   In the MovieLens dataset, the movies are ordered by increasing `movie_id`. With the approach we mentioned above, LeakGAN learnt this ordering through its CNN feature extractor, even though the `movie_id` were encoded with different numbers, not in increasing order, in the training data. We did not want this artifact of the initial dataset to be learnt by our model. To change this behavior we then randomly shuffled the movie histories, before adding movies in the training data.

We also changed the vocabulary size and maximum sequence length. Seeing that the GAN did not take advantage of the encoded movie number sorted by genre, we discarded that step. This time, we built the vocabulary by fixing its size in advance, and we kept only the movies appearing more frequently in the complete dataset. Later we randomly sampled histories of a fixed maximal length and the movie filtering is not specific to this sampling. The previous approach was also slow to train, so we decided to reduce the maximum sequence length to 32 (as in LeakGAN given code) and the vocabulary size to 5 000 words. We added the two previously mentioned special tags: one for movies not in the vocabulary and one for padding. When one tag for out of the vocabulary movies is present in an history, it is followed by padding tags. It means that the original history was longer but the remaining movies were not in the vocabulary and could not be added.

Training

LeakGAN's training procedure first pre-trains the discriminator and the generator, and then it iterates the adversarial training. To mitigate the mode collapse issue, they use "interleaved training", i.e., training one epoch of supervised training for the generator every 15 epochs of adversarial training.

The first time we trained LeakGAN with the MovieLens dataset, we used a vocabulary size of 5847 and a maximum sequence length of 77 as discussed earlier in Section 3.3.2. The model was trained for 44 hours in total using Nvidia GeForce RTX 2070 Super. It performed more than 20 hours of pre-training and then took 36 minutes per epoch. It was stopped after 40 epochs (approx. 24 hours).

The models were mostly run locally, training was performed during the night. Therefore, the training procedure was generally stopped after roughly 12

hours. Training can be resume by restoring the model from saved weights. With this in mind, an optimal training time would be around 12 hours. Therefore, with the second approach, the parameters are changed to a vocabulary size of 5002 and a maximum sequence length of 32. An example run that lasted 12 hours can be breakdown in around 5 hours of pre-training and 7 hours to train 60 epochs. It thus takes 7 minute per epoch. Compared to the 36 min per epoch with the previous parameters we have a 5 times speedup.

The parameters were modified for computational reasons. Different GAN architectures will allow different vocabulary size or maximum sequence length for the same training time. LeakGAN is slow to train partly due to the complexity of its architecture, which allows for less parallelization, since it uses LSTM. There are newer Text GANs which are faster and report better generation capabilities. They also allow to increase the parameter size. In addition, an adversary with more computational power and time can already use bigger parameters with LeakGAN. Recalling our threat model (see Section 2.3), the assumed strong adversary in FLoC is Google (no one else has access to the SimHashes). Google is known for having enormous computational resources. Above that, they also have many talented engineers. Some can build a customized GAN that is faster for sets (we waste a lot of resources training to form the order in sentences). Finally, they have more training data since they operate Chrome and opted-in users provide data in the Chrome User Experience Report. Therefore, there are many reasons for them to have much better results than we have.

The model could have been trained longer as saved weights can be restored. Nevertheless, when we tried to measure the improvements made between the saved models at different epoch, it did not seem necessary to train more for our attack to be successful (discussed in Chapter 4).

Outputs

After training, we manually inspect some produced outputs. When we tried the GAN on the COCO caption dataset, it was rather easy to check the output and see how well it could generate meaningful sentences. With movie histories though, it is harder to check if it seems like a plausible user history or not.

As noticed when trained on COCO Captions, LeakGAN has a tendency to start an history with the same word (e.g., 'A') or movie in our case ('Toy Story'). In addition, the model learnt the movie IDs increasing order from the training data, even when it was not apparent from the encoded movie in the training. It still happened that the order was not respected along the whole sequence, especially for long ones. Some movies appeared multiple times in the output. SimHash takes a set as input, so we removed duplicates in post-processing to fix this issue for our purposes. We also noticed interesting
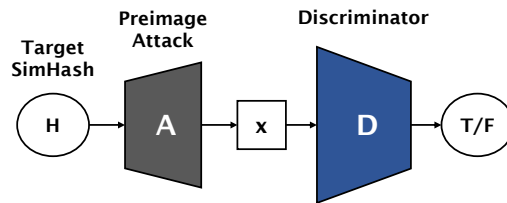
20

Figure 3.1: Pipeline: discriminator applied to the preimage attack outputs

properties which seem natural in some generated histories. Movies part of a franchise would usually be seen in order. Our trained model seems to have learnt that to an extent, we see movies from the same franchise appearing multiple time, though for the order it still does not master it completely. The order in increasing movie IDs is respected but some movies in the film series can be omitted.

In Table 3.1, we colored the titles of multiple movies belonging to the same franchise. We can see that the complete *Harry Potter* movie franchise has been generated along with *The Lord of the Rings*. The first two movies of the *X-Men* original trilogy and the first movie of the *The Chronicles of Narnia* film series are also present. However the *Star Wars* franchise has only 3 movies not all in order.

In Appendix C, we provide more example outputs from the GAN generation:

- One more example with the same GAN parameters, namely a vocabulary size of 5847 and a maximum sequence length of 77.

- An example with smaller GAN parameters (vocabulary size of 5002 and maximum sequence length of 32).

The evaluation of LeakGAN's outputs will be discussed in more detail in Section 4.1.

3.4 Discriminator Applied to the Preimage Attack Output

Our objective is to use the GAN we just trained with our preimage attack to keep only plausible user histories matching a target SimHash. One way to proceed is to use the discriminator of the GAN. It differentiates histories from the preimage attack (described in Section 3.2.2) according to their plausibility. The discriminator outputs a probability, how confident it is about the classification as real or generated. Fig. 3.1 illustrates the pipeline.

Table 3.1: Example output from LeakGAN

TokenID	MovieID	Title	Genres
4216	110	Braveheart (1995)	Action, Drama, War
684	260	Star Wars: Episode IV - A New Hope (1977)	Action, Adventure, Sci-Fi
687	1196	Star Wars: Episode V - The Empire Strikes Back (1980)	Action, Adventure, Sci-Fi
703	3793	X-Men (2000)	Action, Adventure, Sci-Fi
5691	4246	Bridget Jones's Diary (2001)	Comedy, Drama, Romance
189	4896	Harry Potter and the Sorcerer's Stone (2001)	Adventure, Children, Fantasy
164	4993	The Lord of the Rings: The Fellowship of the Ring (2001)	Adventure, Fantasy
165	5816	Harry Potter and the Chamber of Secrets (2002)	Adventure, Fantasy
166	5952	The Lord of the Rings: The Two Towers (2002)	Adventure, Fantasy
1372	6333	X2: X-Men United (2003)	Action, Adventure, Sci-Fi, Thriller
2256	6377	Finding Nemo (2003)	Adventure, Animation, Children, Comedy
5709	6942	Love Actually (2003)	Comedy, Drama, Romance
4284	7153	The Lord of the Rings: The Return of the King (2003)	Action, Adventure, Drama, Fantasy
171	8368	Harry Potter and the Prisoner of Azkaban (2004)	Adventure, Fantasy, IMAX
988	40815	Harry Potter and the Goblet of Fire (2005)	Adventure, Fantasy, Thriller, IMAX
190	41566	The Chronicles of Narnia: The Lion, the Witch and the Wardrobe (2005)	Adventure, Children, Fantasy
3783	50872	Ratatouille (2007)	Animation, Children, Drama
4009	54001	Harry Potter and the Order of the Phoenix (2007)	Adventure, Drama, Fantasy, IMAX
1414	56174	I Am Legend (2007)	Action, Horror, Sci-Fi, Thriller, IMAX
5745	56367	Juno (2007)	Comedy, Drama, Romance
4702	63082	Slumdog Millionaire (2008)	Crime, Drama, Romance
743	68358	Star Trek (2009)	Action, Adventure, Sci-Fi, IMAX
3994	68954	Up (2009)	Adventure, Animation, Children, Drama
878	69844	Harry Potter and the Half-Blood Prince (2009)	Adventure, Fantasy, Mystery, Romance, IMAX
745	72998	Avatar (2009)	Action, Adventure, Sci-Fi, IMAX
4862	74458	Shutter Island (2010)	Drama, Mystery, Thriller
5202	79132	Inception (2010)	Action, Crime, Drama, Mystery, Sci-Fi, Thriller, IMAX
4806	81591	Black Swan (2010)	Drama, Thriller
669	81834	Harry Potter and the Deathly Hallows: Part 1 (2010)	Action, Adventure, Fantasy, IMAX
4291	88125	Harry Potter and the Deathly Hallows: Part 2 (2011)	Action, Adventure, Drama, Fantasy, Mystery, IMAX
5476	92259	Intouchables (2011)	Comedy, Drama
4560	96821	The Perks of Being a Wallflower (2012)	Drama, Romance
1178	102903	Now You See Me (2013)	Crime, Mystery, Thriller
4834	116797	The Imitation Game (2014)	Drama, Thriller, War
777	122886	Star Wars: Episode VII - The Force Awakens (2015)	Action, Adventure, Fantasy, Sci-Fi, IMAX
4022	134130	The Martian (2015)	Adventure, Drama, Sci-Fi
5556	134853	Inside Out (2015)	Adventure, Animation, Children, Comedy, Drama, Fantasy

3.4.1 Benchmark

Settings

We measure the performance of this attack by varying the SimHash prefix length. We then measure the time it takes for the preimage attack (Listing 1) to find a history with the target SimHash and the discriminator to classify this history. For each bit length we will run the preimage attack a 100 times. Those 100 runs will be evenly split on 4 seeds used to generate a target SimHash from a real user history in the test data. Since the preimage attack might not terminate, we set a timeout (chosen as 6 seconds after some trials) to stop the process and restart it. The elapsed time divided by the timeout is then the number of restarts.

Note that the attack is implemented in standard Python while the discriminator uses TensorFlow GPU-accelerated library. We also only measure the runtime of the attack itself and the discriminator classification. The preprocessing necessary to have the right input format is not accounted for.

Results

From the results we present, the main takeaway is the mediocre performance of the discriminator to distinguish random data from real data.

We only show the runtime for SimHash prefix of 5, 10, 15, and 20 bits, since we showed that more than 21 bits are unnecessary with the proposed FLoC implementation (Eq. (3.1)). In addition, the preimage attack is already quite slow when it needs to match 20 bits.

In Table 3.2, the confidence shows the average and standard deviation of the discriminator's output probability for the positive label (classified as real data) when it was the one returned (probability greater than 0.5). Note that the discriminator only classified something as generated (negative label) twice, once for the 15 and 20 bit length (with 77 and 82% confidence). Therefore, it seems that we cannot trust the discriminator to distinguish between generated and real histories when we randomly sample movies to form a history.

We also report the total time for all 100 iterations, and in parenthesis the average time and standard deviation. The runtime of the discriminator is negligible, since one iteration typically takes only 5 ms.

We see that the attack time increases exponentially with the bit length. Namely, we interpolated those four points in Fig. 3.2, and found a function $f(x) = e^{\frac{x-5}{3}} + 4$, where x is the bitlength. Despite this being just an approximation, the exponential nature of the increase in runtime was also expected.

23

Table 3.2: Benchmark of Discriminator – Preimage Attack

Bit Length	Attack Time	Discriminator Confidence
5	497.1 s (5.0 ± 0.6 s)	$99.5 \pm 3.1\%$
10	747.1 s (7.5 ± 4.8 s)	$99.9 \pm 0.7\%$
15	2 258.7 s (22.6 ± 20.4 s)	$99.1 \pm 4.7\%$
20	19 125.5 s (191.3 ± 276.7 s)	$99.6 \pm 3.5\%$
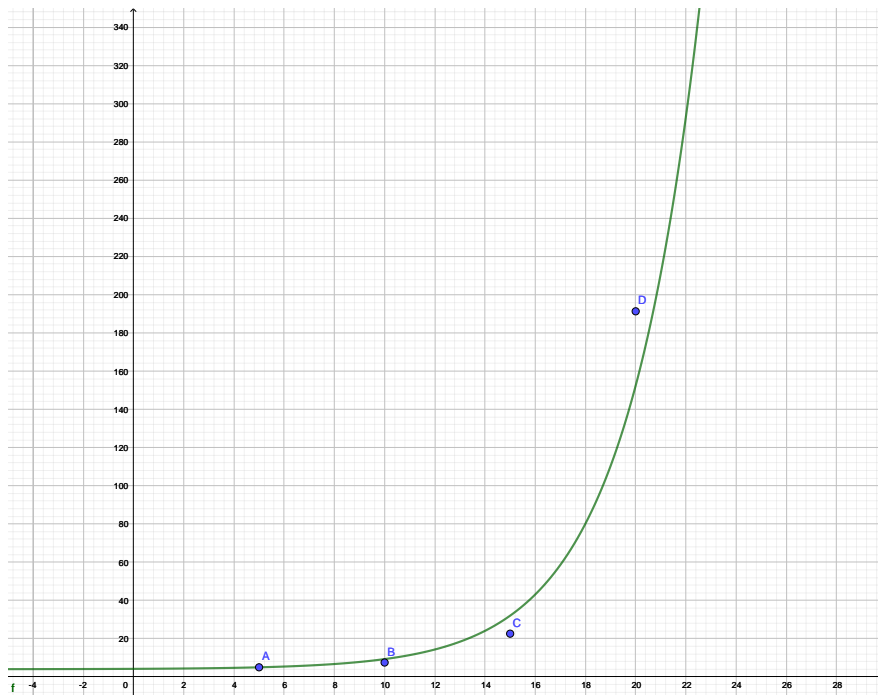
Figure 3.2: Average Preimage Attack Time is Exponential

The preimage attack works, but it is inefficient for higher bitlengths, even after reducing the domain from which we randomly sample movies by the vocabulary of the trained GAN.

However, the speed is not the main limitation that we observed about our proposed history generation. We observed a high level of confidence in accepting the generated histories by the discriminator as real. In fact, we observed only two histories that were ever rejected as generated. We tested if this is influenced by the generated histories' length, but we were rarely able to detect the discriminator rejecting any history. This unfortunately means the idea of filtering generated data from the preimage attack by their plausibility is not working as intended. This is most likely a problem related to LeakGAN or its training. Some other GAN's discriminator should be able to perform better with adequate training.

However, we saw earlier (Section 3.3.2) that the generator still produced decent movie histories. It could be partly explained by the features from the discriminator being leaked to the generator as additional feedback, which is more informative than the resulting classification. The observation that the generator produces reasonable data and that it is much faster than the preimage attack leads to an alternative approach that we describe in the next section.

3.5 Integer Programming on the Generated Data

Our trained discriminator network was unable to distinguish between real and fake history. In this section, we describe how to postprocess a history from the generator network such that it matches a target SimHash. We used integer programming to do this and it significantly outperforms the preimage attack proposed in Section 3.2.

We can solve the SimHash constraints for a target value using the GAN generator's output. We want to find the largest subset of the generated history that would collide with the target SimHash.

3.5.1 Defining the Integer Program

We give below the *canonical form* of an integer linear program

$$
\begin{aligned}
&\text{maximize} && \mathbf{c}^{\mathsf{T}}\mathbf{x} \\
&\text{subject to} && A\mathbf{x} \le \mathbf{b}, \\
& && \mathbf{x} \ge \mathbf{0}, \\
&\text{and} && \mathbf{x} \in \mathbb{Z}^n,
\end{aligned}
$$

The vector $\mathbf{x}$ is the vector we are looking for. For SimHash we only need a binary integer program, meaning the elements of $\mathbf{x}$ are either 0 or 1. A 0 coordinate in $\mathbf{x}$ means that the movie at this position is not part of the subset matching the target SimHash. We want to maximize the size of the subset. The sum of the entries in $\mathbf{x}$ gives us the size of the subset and we want to maximize this objective. We therefore set $\mathbf{c}$ to be the vector with only 1s. A bit in a SimHash is 1 if the sum of Gaussian samples is greater than 0, otherwise the bit is 0. Hence $\mathbf{b}$ is the zero vector and the inequality that defines the k-th bit value of the output SimHash can be written as

$$
\sum_{i=0}^{n-1} a_{k,i} \cdot x_i \le 0
$$

where $a_{k,i}$ is an element of A and x_i of $\mathbf{x}$. A is an m by n matrix where m is the SimHash bit length and n is the length of our input history. The matrix

25

A will contain the Gaussian coordinates from the SimHash computation. The Gaussian coordinates can be computed as they depend on the SimHash output bit position and the current hash. We define

$$f(\text{bit}, \text{hash}) = \text{random\_gaussian}(\text{bit}, \text{hash}) \cdot \text{sign}(\text{bit}),$$

where $\text{sign}(\text{bit})$ is 1 if bit is 1 and -1 if bit is 0. Now we can write

$$A = \begin{pmatrix} f(0, \text{hash}_0) & f(0, \text{hash}_1) & \cdots & f(0, \text{hash}_{n-1}) \\ f(1, \text{hash}_0) & f(1, \text{hash}_1) & \cdots & f(1, \text{hash}_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ f(m-1, \text{hash}_0) & f(m-1, \text{hash}_1) & \cdots & f(m-1, \text{hash}_{n-1}) \end{pmatrix}$$

We can rewrite our integer linear program as follow:

$$
\begin{aligned}
& \text{maximize} && \sum_{i=0}^{n-1} x_i \\
& \text{subject to} && A\mathbf{x} \leq \mathbf{0}, \\
& && \mathbf{x} \geq \mathbf{0}, \\
& \text{and} && \mathbf{x} \in \{0,1\}^n.
\end{aligned}
$$

0 is always a trivial solution to the integer program. It is another reason for the use of a maximum in the objective. So a valid solution for our purpose is one where the solution of the integer program is greater than 0.

3.5.2 Applying the Integer Program

The integer program can be translated into Python code straightforwardly with the use of a library. We first tried the library `cvxpy` as it was easier to start with than the more powerful `gurobi` solver. Nevertheless, both of those solvers do not allow for strict inequalities. Therefore, we cannot reproduce exactly the SimHash constraints. In Chromium's SimHash implementation the bit is 1 if the sum of Gaussian samples is greater than 0 and consequently the bit is 0 if that sum is less or equal to 0. Thus, in the case where we have equality with 0 we might infer the wrong bit with our integer program.

It is hard for a linear program solver to optimize strict inequality constraints. `cvxpy` documentation even says that "Strict inequalities don't make sense in a real world setting"[2]. From a theoretical point of view, with strict inequality, the space of feasible solutions becomes an open set. Finding extrema on an open set is not desirable as one can get closer and closer to the boundary but not reach it. In practice, with finite floating-point arithmetic, a

---

[2]https://www.cvxpy.org/tutorial/intro/index.html#constraints

computer cannot differentiate excessively close numbers. Therefore, solvers use a feasibility tolerance and the difference between strict and non-strict inequality is not substantial.

One might say that the equality with 0 case in SimHash constraints should not appear often. However, when we later try to find all the solutions to the same SimHash problem with `gurobi`, it will most likely also find solutions that do not match the target SimHash. Some bits are flipped due to the equality with 0.

To remediate this issue, we compute the SimHash of the resulting preimage and ensure it matches the target SimHash.

Finding Multiple Solutions from the Same History

By default an integer program solver would return only one solution satisfying the problem. It could also return the suboptimal solution it found on the way. However, we might want to find more than one solution from the same problem. The objective of our integer program is to return one of the maximum subset of the history that matches the target SimHash, so suboptimal solutions that still satisfy the SimHash constraints but have shorter length histories are skipped. These shorter solutions can more precisely match the history and reduce noise of unnecessary movies.

A first attempt with the `cvxpy` library would rerun the solver with an added new constraint to bound the maximum to a smaller value. Presupposing that the solver returned a valid solution, namely a value greater than 0. This approach is limited by the fact that `cvxpy` only returns one solution per maximal history subset length.

There exists algorithms to find multiple solutions to an Integer Program. Gurobi abstracts those away and can return multiple solutions with their MIP Solver. The solver gives us some degree of freedom to choose how it returns multiple solutions, namely up to 2 billions best solutions[3]. This can cause issue with non-zero optimality tolerance, i.e., those solutions might not be the best possible. However we did not take care of this. We verify the optimality of each solution due to the potential bit flip when the constraint equals 0. Furthermore, we are not concerned with finding the best solutions, we want as many solutions as possible in a limited computation time. We can set a fix number of solutions to find or set it to the maximum and add a time limit on the solver. The solver will try to look for the specified number of best solutions or prove that there are not that many.

This procedure can take time, more time than switching to another problem with a different input history, resulting in an exploration-exploitation

---

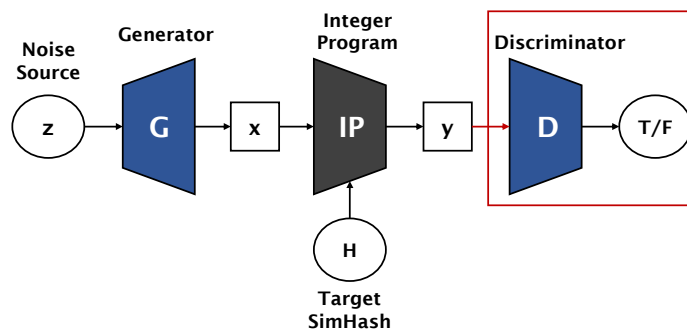[3] https://www.gurobi.com/documentation/9.5/refman/poolsolutions.html

Figure 3.3: Pipeline: integer programming on the generator outputs

dilemma. Should the solver continue to find solutions from the same problem or switch to the next problem. The solution count and time limit parameters can be used to promote more exploration or exploitation. We also have to think about the trade-off between quickly generating many histories and the diversity of those generated history. When we find multiple thousands history matching a target SimHash from the same 32 movie set, those histories will lack diversity. However, optimally the generated history matching a target SimHash is similar to real users' history matching the same SimHash. The more diverse the generated histories are, the more insight we can have about the real users histories.

3.5.3 Benchmark

Setting

We have a pipeline of the form:

$$\text{History Generation} \rightarrow \text{Integer Program} [\rightarrow \text{Discriminator}] \qquad (3.2)$$

where we can include or not the discriminator post-processing step.

We illustrate the updated pipeline in Fig. 3.3. We can optionally (red frame) apply the discriminator on the integer program's output.

Each of the steps in Eq. (3.2) allows configuring fixed or specified parameters. Some of them will vary when we compare different pipelines.

Let us start with the History Generation parameters. We can either use the LeakGAN generator or a Random History generator which randomly samples a matrix of the same size as LeakGAN (batch_size × sequence_length) with tokens from the vocabulary. The batch size (64), vocabulary size (5000) and maximum sequence length (32) are parameters of the GAN that also apply to the Random generator. For the GAN we also have the possibility to load a saved checkpoints to restore the model weights. We explore this functionality in Chapter 4.

The Integer Program parameters depend on the solver (`cvxpy` or `gurobi`). However, once we had `gurobi` working, we only kept this one, since it is a more powerful solver. With the gurobi solver, we can set a time limit and whether the solver should search for multiple solutions. By default, only the solutions found on the way to the optimal one are returned. We can then specify how many solution to look for, a fixed number smaller than the maximum possible.

Now we have to define some stopping criteria according to the target SimHash parameters.

- How many target SimHashes do we want to match.

- How many unique histories we want to match a target SimHash.

- Whether those histories also have to be validated by the discriminator after matching the target hash.

- The sampling domain for the target SimHash (e.g., training data, test data, complete MovieLens dataset).

- The bit range of the target SimHash.

We are measuring the runtime to estimate the speed of our attack. Note that the Python code is not always optimized for speed and logging is likely slowing it even more. We measure it only at the line executing the code of interest to improve precision. For example, we measure the call to optimize the integer program and omit the construction of the integer program. It is in a sense similar to what we have done for our earlier benchmark in Section 3.4.1.

Integer Program on Generator Output

To allow for comparison with the runtimes results from Table 3.2, we try to reproduce similar settings (Section 3.4.1). We thus make the SimHash bit length vary from 5 to 25 (with 5 increments). We also generate 4 different target SimHashes from the history of real users in the test set. The SimHash obviously are different due to the increasing prefix bit length. For each target SimHash, we set the target collisions count to 25. Due to the history generation in batch of 64, we will most likely end up with more than 25 collisions. The discriminator is not used in this pipeline, but we have already seen that the discriminator runtime is negligible.

In Table 3.3, we report, in the 'History Generation' column, the number of histories generated by the GAN, followed by the number of those histories that only admit zero as solution to the integer program. We also report, in the 'Integer Programming Time' column, the total runtime, and in parenthesis the average time and standard deviation. We did not modify the GAN

Table 3.3: Benchmark of GAN – Integer Program

Bit Length	History Generation	Integer Programming Time
5	256, 0	0.52 s (2 ± 1 ms)
10	256, 14	2.01 s (8 ± 10 ms)
15	256, 92	5.03 s (20 ± 19 ms)
20	256, 170	5.89 s (23 ± 23 ms)
25	704, 626	12.83 s (18 ± 22 ms)

parameters while varying the bit length and its runtime is negligible, the generation of a batch of 64 histories typically takes less than 0.5 s. We need to remember that the generator runs in batch of 64 histories, while the integer program is run for each history. This means that the generator is called 4 times while the integer program is called 256 times. Except for the last row (bit length of 25), where 11 calls to the generator and 704 calls to the integer program solver were needed to reach the target collisions count. This explains the increase in runtime for the last row. From the results in Table 3.3, it is clear that incrementing the bit length increases the complexity of the integer program. A higher hash length means more SimHash constraints to satisfy. And with the fixed history length as input, the integer program fails more often to find non-empty history for which a subset can match the target SimHash.

The average solve time is similar for the bit length range 15 to 25. However, the standard deviation indicates that we have a lot of fluctuations between individual integer programs. Every integer program admits 0 as a solution. It generally requires more time to solve the integer program when 0 is not the only solution. Nevertheless, 0 is often the only solution. Undoubtedly, the lower the hash length the easier it is to find non zero solution. The integer programming problems with only trivial solutions bring the average down, while the others increase the standard deviation. Let us take the 25 bit length as an example. We have to remember that by default `gurobi` returns more than one solution, if it found more while searching for the optimal one. Hence, for the 25 bits case, 626 histories did not satisfy the SimHash constraints with a nonzero solution. From the 78 histories that could, the solver returned 144 collisions. Therefore, we can see that with higher bit lengths it is harder to find non-zero solutions.

This is also illustrated with the average length of generated histories. While the average length of an history generated by the GAN is stable around 27 ± 4. The average length of an history after the integer program is around 15 ± 5. Obviously for the integer program output, the longer the bit length is, the lower the average and standard deviation are. For a bit length of 25 the average history length is 13.8 ± 3.7 when for 15 it is 15.3 ± 4.5.

If in future work we wanted to accomodate longer SimHash bit lengths, we could simply increase the maximum history length and train a GAN with the updated maximum sequence length. A bigger vocabulary size can also help. However, as we already argued (Eq. (3.1)), for our purposes a bit length of more than 22 is disproportionate.

Comparing the results with Table 3.2, we see tremendous improvements to the speed for higher bit lengths. The runtimes went from thousands of seconds to tens of seconds. We have to keep in mind that integer programming is NP-complete [31]. Therefore for higher hash and history lengths the runtime should increase exponentially. However, the integer program does not need to randomly restart when it is in non optimal territory. It needs new histories (possibly of higher length), but not an increase of the domain hash (5000 movies). This makes it more reliable than the the previous preimage attack.

With this experiment, we can be confident that our attack can accommodate the various SimHash prefix lengths used in the FLoC computation.

Integer Program Explicit Search for more Solutions

We have already seen that `gurobi` can return a various amount of solutions if it found them before the optimal solution. However, we want to know how much we can generate and how fast, if we were to let the solver explicitly look for more solutions.

Given the specified size of cohorts (2000) during FLoC origin trial (discussed in Section 2.2.2). It seems excessive to ask the solver to look for more solutions, when one good solution can already give us thousands to hundred thousands histories matching a target SimHash. The latter being tested when the full history already matches the target SimHash. For example, with a 20 bit SimHash and a time limit of 100 s, the solver found 168 745 solutions and more could still be found.

For this experiment we set the SimHash bit length to 20. We set the number of solutions to search to the maximum possible value. Either the cardinality of the power set of the history or the maximum value allowed by `gurobi` (2 billions). We only vary the time limit, as it can take a lot of time for the solver to prove that there are no more solutions even for problem with a small number of solutions. If we were not interested in generating many histories in a short time, we could set a lower maximum number of solutions to search. We aim to generate 2000 histories matching a target SimHash. As mentioned earlier, it was the minimum cohort sizes during FLoC's origin trial. We only match one target SimHash. We do not apply the discriminator on the outputs.

We report in Table 3.4 the total runtime of the integer program, and in parenthesis the average time and standard deviation ('I.P. Time' column). We omit

Table 3.4: Benchmark of Generation $\rightarrow$ Integer Program Pipelines

Pipeline	I.P. Time	Solutions	Counts (G,IP)
GAN	100.20s (0.02 ± 0.02s)	2037	4992, 1090
GAN Multisol 10	2.84s (0.04 ± 0.06s)	6242	64, 19
GAN Multisol 20	3.23s (0.05 ± 0.07s)	2615	64, 15
GAN Multisol 30	3.60s (0.06 ± 0.11s)	6608	64, 14

the one for the generator as it does not change with the current parameters' variation. We also report the number of solutions found ('Solutions' column). The 'Counts' column shows how many histories had to be generated to reach the target solution count, followed by the number of histories that admitted a non-empty subset matching the target SimHash. The pipelines include a GAN which is not set to search for other solutions than the one that leads to the optimal solution. GAN Multisol 10 is set to search for multiple solutions, possibly all of them with a time limit of 10.

From Table 3.4 we can see that for our purposes searching for more solutions is excessive, if we want to flood a cohort with only 2 000 users. We would get more diversity if we generate more histories. On the other hand, GAN Multisol is quite fast. Compared to the 100 seconds and 5 000 generated histories needed to reach the target solution counts. The GAN Multisol can generate thousands of solutions with less than 20 histories.

Unfortunately, this particular run or experiment does not showcase the use of increased time limits, since the solver finds all solutions in no time. The variation in the GAN history generation makes it hard for the integer program to find a lot of solutions. Improvements could be made on the generation part in future work.

We can mention that some solutions are discarded after checking if they match the target SimHash. Due to the equality with 0 constraints being not respected. It is the case for 16 solutions of GAN Multisol 10.

Larger cohort sizes would have been used if FLoC was tested outside a trial with a very small percentage of Chrome users. But this is not a problem as our attack is again successful, even for cohort much larger than 2 000. We also note that if we sacrifice diversity in the generated histories, we can drastically increase the speed of the attack.

Discriminator applied to Output of Integer Program

We experimented with the following pipeline:

History Generator $\rightarrow$ Integer Program $\rightarrow$ Discriminator

Table 3.5: Discriminator Confidence depending on Generation Method

Generator	Real Confidence (samples)	Gen. Confidence (samples)	Counts (Gen, IP)
RAND	$99.6 \pm 2.1\%$ (2481)	$72.7 \pm 15.5\%$ (8)	1284, 1127
GAN	$94.9 \pm 10.3\%$ (1900)	$88.4 \pm 14.4\%$ (624)	1728, 1236

The random generation and discriminator runtimes are very fast, much faster than the LeakGAN generation. We can note that with random data the integer program has an easier time finding preimages. It is most likely due to the random data having more diversity in the included movies. However, as we will confirm below with Table 3.5, the discriminator seems to not be trustworthy when applied to random data. We already saw (Table 3.2) it is overconfident on unseen data being real data. Meaning that if we care about our preimages to resemble real user data, we might not want to use this pipeline.

For Table 3.5, we ran the experiment with 10 different target SimHashes. Those 15 bits SimHashes are generated from real users history in the test data. For each target SimHash, at least 150 collisions have to be generated. Those collisions have to be classified as real by the discriminator. We only vary the generator to be either LeakGAN or random.

In Table 3.5, we report the confidence probability for the label (real or generated) given by the discriminator. As a reminder, the confidence shows the average and standard deviation of the discriminator's output probability for a label when it is the one returned (probability greater than 0.5). We also give the number of samples that were classified as real or generated. The counts refer to the number of generated histories (multiple of the batch size i.e. 64) followed by the subset that admitted a non-trivial solution to the integer program ('Counts' column). The number of samples that the discriminator classified is more than double the number of histories that admit a solution to the integer program (e.g. $2489 > 2 \times 1127$). This is because the integer program also returns suboptimal solutions found on the path to the optimal solution. Therefore, it can return more than 1 solutions for one problem.

The solutions from the integer program contained a comparable average history length (env. 16) between the GAN and Random generation. Yet the discriminator classified the random histories subsets matching a target SimHash almost entirely as real (2481 out of 2489). On the other hand, the discriminator seemed to be able to classify around one fourth of the GAN's generator history subsets as generated (624 out of 2524). So it seems that

the discriminator labels data from unseen distribution (random generation) as real by default. Whereas, on the data it trained with, it still manages to classify some of them as generated. This confirms our assumption that random histories do not resemble the training data and the discriminator somehow classifies those samples as positive. Therefore this also makes the pipeline with the discriminator applied to random data less useful for our purposes. As this data would not resemble user data as much as the one generated by the GAN. This is unfortunate as the random generation is very fast.

With this experiment, we notice that we can use the GAN to generate plausible histories. The generated histories are still classified as real with high probability by the discriminator, even after being piped into the integer program to find a subset matching a target SimHash. Therefore, our attack pipeline (GAN with integer program) succeeds in breaking FLoC's k-anonymity property. Since our attack allows reverse-engineering the browsing history from users' SimHash, the privacy of FLoC is compromised.

Chapter 4

Evaluation

Metrics that are representing the performance of our attack well can also be used to evaluate the GAN during training, namely as an early stopping criteria.

4.1 Movie History Generation

In LeakGAN's paper [27], they reported BLEU scores for realistic text generation benchmarks. BLEU (bilingual evaluation understudy) [47] evaluates the quality of text that has been translated from one language to another without human assistance. However, that is a metric used in NLP, which is not suitable for our attack since it is sensitive to the word order. Indeed the BLEU metric reported in LeakGAN's evaluation uses n-grams (for n in $\{2, 3, 4, 5\}$). An n-gram is a contiguous sequence of n words from a sentence. The generated sentences are compared (using set of n-grams) to reference sentences in the test set. For each generated sentence a BLEU score is computed. It can take values between 0 and 1 and it shows how similar the generated sentence is to the reference sentences, higher scores suggest more similar sentences. The scores are then averaged to estimate the overall quality. This metrics when applied to our movie history generation gives very poor results (close to 0) for longer n-grams as we tried to train our GAN so as to avoid learning an ordering.

We wanted to report different results than the BLEU scores to evaluate the outputs of the GAN. We opted for metrics that are not order-sensitive and can evaluate how similar generated histories are to the underlying dataset based on real user histories.

Such metrics can serve model evaluation (e.g., early stopping of training), but also evaluation of the complete process or its components (e.g., data generated by the preimage attack).

4.1.1 Common Movies

We considered several metrics for evaluating different set of watched movies (that we call histories). For example, for movie franchises, one can check if different movies of the same franchise appear and follow some sort of chronological order (by release date or chronological event in the story). We did observe that some histories generated by the GAN exhibit such properties, while others do not (see Section 3.3.2). However, it is preferable if the metric is more generic, as not every history contains a movie franchise, and finding an equivalent information for websites would be complicated. We settled for a metric that counts the number of common movies.

In the context of FLoC the common movies become common domains. And if the generated histories share a lot of domains with unseen real user's browsing histories, then we succeeded in leaking potentially sensitive information about the target user.

Settings

To compare the performance of the GAN during training, our metric needs to be fast and must not require excessive usage of data. Runtimes measurements have already been done in the previous chapter. We refer to Section 3.5.3 for more details on the pipeline possible parameters. As we use the same program to evaluate our pipelines here.

As the input of the metric we use real and generated histories. Namely, 5 invariant real user's histories from the test set serve as goal. To these, we match 200 histories generated by an integer program searching for 15 bits prefix collision. We can then apply this metric on various GAN models in the intermediate stage: after the generator, the integer program, and the discriminator. As reference, we can use a random generator instead of the GAN's generator.

Results

We compute the number of common movies between the generated histories and five unseen real user histories used to compute the target SimHash. We want our attack to generate histories that will have common websites (here movies) with some target users in the same cohort. Therefore, computing the number of shared movies between real and generated history is an appropriate validation method for that purpose.

In Table 4.1, the results are aggregated over the generation for the five target SimHashes. For each target, we computed the average of the common movie counts weighted by the number of history with those counts. We report the mean and standard deviation of the list of averages under the 'Common

Table 4.1: Distribution of Common Movie Counts

Generator	Common Movies		Counts	
	Generator	Int. Prog.	Gen.	I.P.
RAND	0.17 ± 0.04	0.11 ± 0.04	16.20 ± 2.14	11.80 ± 2.9
GAN-11	2.34 ± 0.97	1.58 ± 0.81	25.80 ± 2.64	23.2 ± 3.19
GAN-21	2.10 ± 0.84	1.46 ± 0.76	24.00 ± 3.95	22.00 ± 3.58
GAN-31	1.90 ± 0.68	1.30 ± 0.61	$\mathbf{27.60 \pm 2.58}$	$\mathbf{26.00 \pm 2.76}$
GAN-41	$\mathbf{2.42 \pm 1.04}$	$\mathbf{1.67 \pm 0.93}$	25.00 ± 2.53	22.60 ± 2.73
GAN-51	2.01 ± 0.76	1.39 ± 0.64	26.40 ± 2.58	23.6 ± 3.56
GAN-61	1.94 ± 0.68	1.29 ± 0.61	26.40 ± 3.01	23.40 ± 3.01

Movies' columns. We also report, under the 'Counts' columns, the number of movies from the real user history (max. 32) that also appeared in any of the generated histories. For the GAN's generators the average history length is around 27 and around 15 after the integer program. For the random generator the numbers are 32 and 17. This sets the upper bound on the number of common movies, since the histories filtered by the integer program are only about half of the maximal length.

We omit the results after the discriminator's classifications, as those are very close to the results evaluated after the integer program. They are almost identical for the random generation ('RAND' row in table) and usually get slightly worse within a $5 - 10\%$ margin for the GAN's generation.

From Table 4.1 we see that GAN-41 has the best average number of common movies with real users. But it is GAN-31 which, across all histories, generated the most movies present in the target histories. Note the high standard deviation, so our results might not be robust.

In Table 4.2, we report more statistics about the run that generated the results in Table 4.1. We present the number of histories produced by the generator (column 'Generated'), the one that admit a nontrivial solution to the integer program (column 'Int. Prog. Success') and the number of solution returned (column 'Solutions'). We also report the total number of movie present in the vocabulary that were generated (column 'Unique Movies').

We see that the random generator is the one that needs the least samples to match the target collision counts. It also has no problem generating diverse movie outputs as it randomly samples in the movie vocabulary. The GAN models vary based on training checkpoints. The model GAN-31 generated the most unique movies, but it could be explained by the fact it also generated a lot more histories. This could also explain why it had the best numbers in Table 4.1's 'Counts' columns. It may also be that GAN-31 had to generate more histories because its discriminator rejected more of them.

Table 4.2: Statistics on the History Generation

Generator	Generated	Int. Prog. Success	Solutions	Unique Movies
RAND	640	548 (85.6%)	1208	4909
GAN-11	1472	993 (67.5%)	1929	2291
GAN-21	1024	738 (72.1%)	1444	2224
GAN-31	1600	1149 (71.8%)	2311	2778
GAN-41	1216	864 (71.1%)	1731	2140
GAN-51	1280	895 (69.9%)	1803	2463
GAN-61	1024	757 (73.9%)	1525	2438

Table 4.3: Confidence of the discriminator

Generator	Confidence Real	Confidence Generated	Counts (Real, Generated)
RAND	$99.57 \pm 2.53\%$	$71.84 \pm 12.62\%$	1203, 5
GAN-11	$90.42 \pm 13.23\%$	$85.52 \pm 15.07\%$	1209, 720
GAN-21	$93.42 \pm 11.50\%$	$83.57 \pm 15.90\%$	1191, 253
GAN-31	$89.57 \pm 13.72\%$	$90.31 \pm 13.44\%$	1111, 1200
GAN-41	$92.08 \pm 11.97\%$	$85.54 \pm 14.97\%$	1196, 535
GAN-51	$93.66 \pm 11.16\%$	$89.07 \pm 14.34\%$	1156, 647
GAN-61	$94.88 \pm 10.58\%$	$88.94 \pm 13.88\%$	1127, 398

For the GAN models, the ratio between the number in the 'Unique Movies' and 'Generated' columns is the highest for GAN-61 at around 2.4 while the second highest is GAN-21 at approximately 2.2. However, those variation may be due to the high variance induced by the training algorithm.

We also compare the confidence of the discriminator. We saw earlier (Section 3.5.3) that it did not help to distinguish random data from real. But it is better in distinguishing GAN's generator data from training data. In Table 4.3 we report the confidence of the decisions made along with the label counts. As a reminder, the confidence shows the average and standard deviation of the discriminator's output probability for the positive label (classified as real data) when it was the one returned (probability greater than 0.5). Note that when the 'Generator' column varies, so does the discriminator since the GAN contains both network. For the random generator ('RAND' row), we used the discriminator of GAN-61.

The confidence of the GAN's discriminator, with weights restored from different checkpoints, seems rather stable. However the different label counts can vary quite a bit, questioning the stability of the discriminator during training. Indeed, GAN-31 generated the most histories (see Table 4.2) because its discriminator rejected more histories than it accepted and it is the only generator for which it happened. Its number of histories classified as
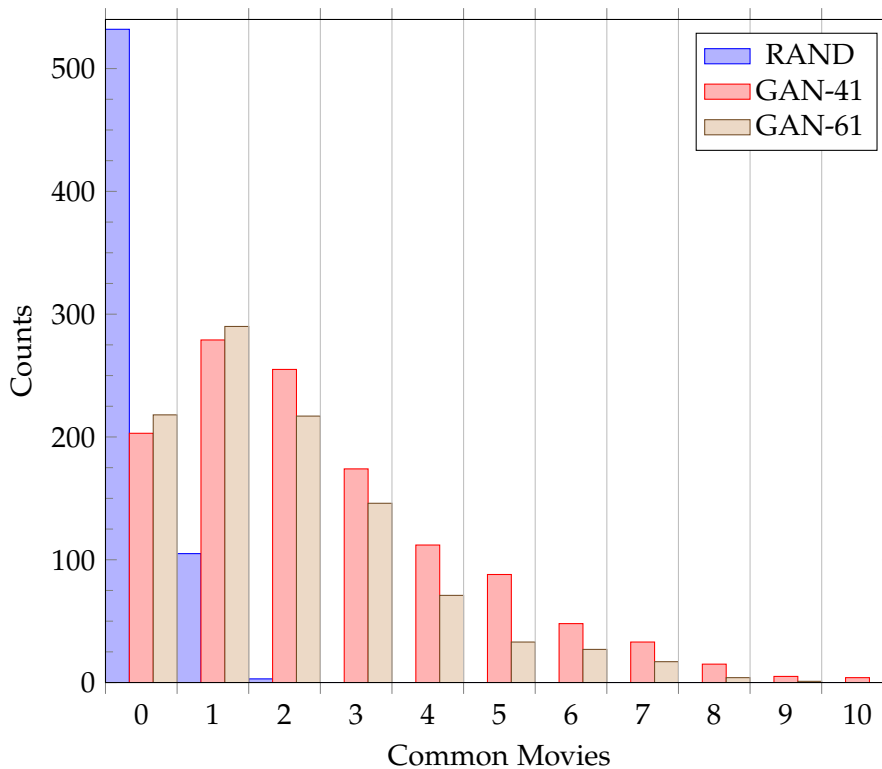
Figure 4.1: Distribution of common movie counts

generated (rejected) by the discriminator is close to double the other model with the highest rejection (GAN-11).

In Fig. 4.1, we show the distribution of the common movie counts between data generated by the GAN's generator and the target SimHashes, aggregated over 5 different target SimHashes. We used only the generator, no integer programming. We also computed the histogram after the integer program and discriminator, and each time the number of common movies decreased slightly, the weights shifted towards the lower numbers.

From the histogram we clearly see that the random generation has very little common movies with the target history. However, our GAN model evaluated at two different checkpoints has a lot more histories with higher number of common movies. This is promising but only the tail of the distribution is on the higher counts, with a maximum of 10 for 4 histories of GAN-41. For GAN-61 the maximum is 9 for 1 history while GAN-41 has 5 histories with 9 common movies with the target history. On average the number of common movies with a target history is around 2 (see Table 4.1).

Table 4.4: Distribution of minimum Hamming distances

Generator	Minimum Hamming Dist.
RAND	17.94 ± 0.10
GAN-11	15.67 ± 0.06
GAN-21	15.85 ± 0.10
GAN-31	16.24 ± 0.06
GAN-41	$\mathbf{15.26 \pm 0.14}$
GAN-51	15.97 ± 0.09
GAN-61	16.36 ± 0.13

4.1.2 Minimum Hamming Distance

Despite FLoC using only the leading 20 bits of SimHash, the Google server responsible for the cohort assignment has access to the full 64 bit SimHash. We measure the Hamming distance of the full SimHash as a metric of dissimilarity of histories. The Hamming distance checks for each bit position if they match and returns the mismatch count between the two hashes.

We compute the minimum Hamming distance among each SimHashes in the test set (5000) and the SimHash of a history just generated by the GAN. In Table 4.4, we present the results. We report the mean and standard deviation after the generation in the same way as in Table 4.1 for the common movies.

Lower Hamming distance is better, since the SimHashes are more similar and similarity of SimHashes should also represent similarity of their inputs, the histories. Histories from the GAN-41 are having the lowest Hamming distance from the target, this model is performing the best and it was also the case with the common movie count (see Table 4.1). We note a certain correlation between the common movies and minimum Hamming distance metrics. We also observed that generally a close Hamming distance between SimHashes corresponds to more common movies. GAN-61 is the worse after RAND, but just a bit higher than what is in the variance of similar Generator. What is more surprising is that GAN-41 is considerably lower than every other Generator, making it a clear winner with respect to this metric.

In Fig. 4.2, we plot the distributions for the same three generation processes.

4.1.3 Wasserstein Distance

The Wasserstein distance (or earth mover's distance) between two distributions g and r represents intuitively the minimum quantity one has to move to convert g into r. If we see the distributions as piles of dirt, we compute the previous quantity by multiplying the chunks of dirt to move with the distance they shifted.
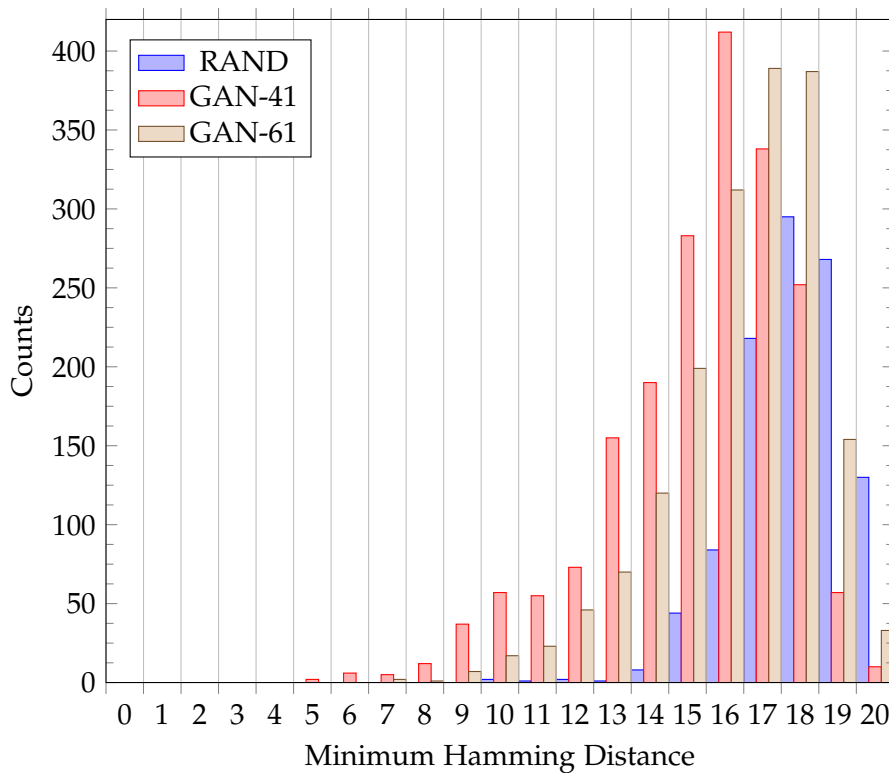
Figure 4.2: Distribution of minimum Hamming distances

Settings

We use the 5000 histories from the test set as reference distribution. We generate around 5000 histories with the GAN generator as the distribution for evaluation. We use a function that compute the Wasserstein distance for one dimensional distributions. To transform the histories into one dimensional distributions, we associate with each possible movie ID (and special padding tags) its number of occurrences in the history list.

Results

In Table 4.5, we report the results varying the model checkpoints and adding the random generation as a baseline.

For this metric, GAN-41 run performed worse than GAN-61, despite some previous metrics showing the opposite. The random generation performs badly as it puts the same weights for every movie in the vocabulary. However, in real users histories not every movie is equally likely to be present.

Table 4.5: Wasserstein distance between generated and real histories.

Generator	Wasserstein distance
RAND	921
GAN-11	473
GAN-21	318
GAN-31	282
GAN-41	367
GAN-51	354
GAN-61	**262**

4.2 k-anonymity

4.2.1 FLoC Whitepaper Evaluation

This metric is taken directly from the FLoC whitepaper [50, Fig. 4]. Their work evaluates the "utility of cohorts for varying levels of anonymity" using cosine similarity of vectors representing the history. We start by reproducing Fig. 4b shown in Fig. 4.3.

When replicating the method with histories generated by our GAN, we evaluate the similarity of the generated histories that belong to the same cohort. We also compare generated and real histories in the same cohort. This allow us to further validate the output from our GAN.

Method description

More details about the feature extraction and cluster assignment used for this method can be found in Appendix D.

In the FLoC whitepaper, authors measure the "utility of cohorts for varying levels of anonymity" by computing the average cosine similarity (normalized dot product) between every history in the cluster and the centroid of that cluster. In our case a cluster is a cohort.

FLoC whitepaper does not specify, which norm was used for the cosine similarity computation. We took the L2 norm for vectors. While experimenting with other norms (vector and matrix) we could get similar or quite different results.

They then computed, as we interpret it, a weighted 2-percentile over the distribution of average cosine similarities for each choice of target cluster size. This distribution is weighted by the respective size of each cluster. For example, say we want 10 cluster of size 500, with a SimHash clustering (see Appendix D.3) we might have clusters with bigger or smaller size than the expected target of 500.
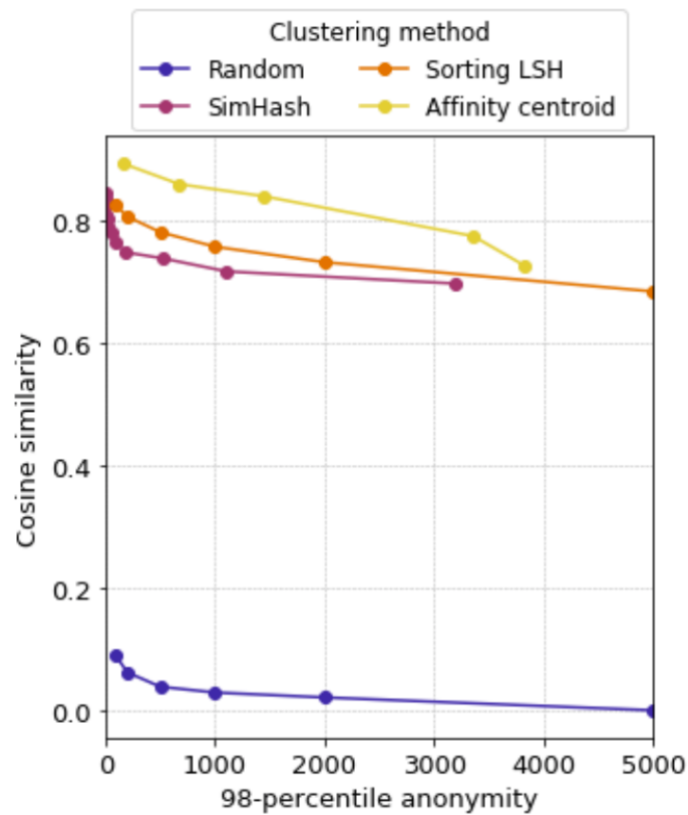
Figure 4.3: Fig 4b. in FLoC whitepaper

This is a way to evaluate privacy, more precisely the *k*-anonymity level of protection that 98% of the user would benefit from. The results from FLoC whitepaper are in Fig. 4.3. With this metric, a result closer to 1 is better.

Results

We tried to reproduce the whitepaper evaluation. We had to make more decisions on how to reproduce the evaluation results from FLoC whitepaper. We discuss these in Appendix D.4.1 and provide alternative results based on different decisions in Appendix D.4.2.

LeakGAN generated enough user for a cluster in around 30 minutes, depending on the cluster size parameters. We generated 9 clusters that we use for all the different results.

The plot in Fig. 4.4 shows our reproduction of the whitepaper evaluation. We tried to reproduce the procedure, inferring the missing parts as was detailed earlier and in Appendix D. We stored several models from different iteration of the training loop, but displaying a plot for each one would not be very

meaningful due to high variation. We hence report the last model – GAN-61 (Figs. 4.4b and 4.4d) and the model with one of the best curve – GAN-41 (Figs. 4.4a and 4.4c). Among the other models, we noticed better and worse curve fluctuating with longer training. There was no clear improvement the longer we trained.

The first row (Figs. 4.4a and 4.4b) is on the full dataset. It makes use of the ratings in the feature extraction for the cosine similarity computations. As mentioned before, GAN does not have access to this information. The second row (Figs. 4.4c and 4.4d) shows the results on the training data subset and without ratings.

On the same row only the GAN curves change. In contrast, the GAN curves are the same in each column since the GAN does not make use of the full input data. However, on the training data the other SimHash algorithms perform worse as their curves go down. There are less users in the training data (120 000 vs 162 541) and their histories are shorter (max 32 vs average 153). Only 5000 movies are retained in the training (more than 60 000 are in the dataset). In addition, no ratings are available for the training data. This can explain the change in the data point positions for the SimHash based clustering using the training data. For more details on the construction of the training dataset, refer to Section 3.3.2.

The random cluster assignment serves as a baseline. Our GAN performs better than random and worse than StrSimHash. StrSimHash is the SimHash used in Chromium, it takes strings (e.g., movie titles) as input, while the other SimHash uses the feature extraction (see Appendix D.2).

4.2.2 Closer Look at Pairwise Cosine Similarity Matrices

We keep the cluster assignments from the previous section. Let us take a closer look at the pairwise cosine similarity matrices between users in the same cluster (same SimHash bit length and value). The Chromium SimHash is used for each cluster we compare. However, some users are real users, while the others are generated by LeakGAN.

Settings

We take the pairwise cosine similarity matrix between the real and generated movie histories. To make the comparison more fair with the GAN generated cluster we use the training data for the Chromium SimHash clusters.

The pairwise cosine similarity matrices can be quite large (up to around 5000x5000) and plentiful ($\approx 2^{10}$ matrices depending on the hash length). We therefore compare the minimum and maximum entries according to selected statistics.

(a) GAN-41 on dataset with genres.

(b) GAN-61 on dataset with genres.
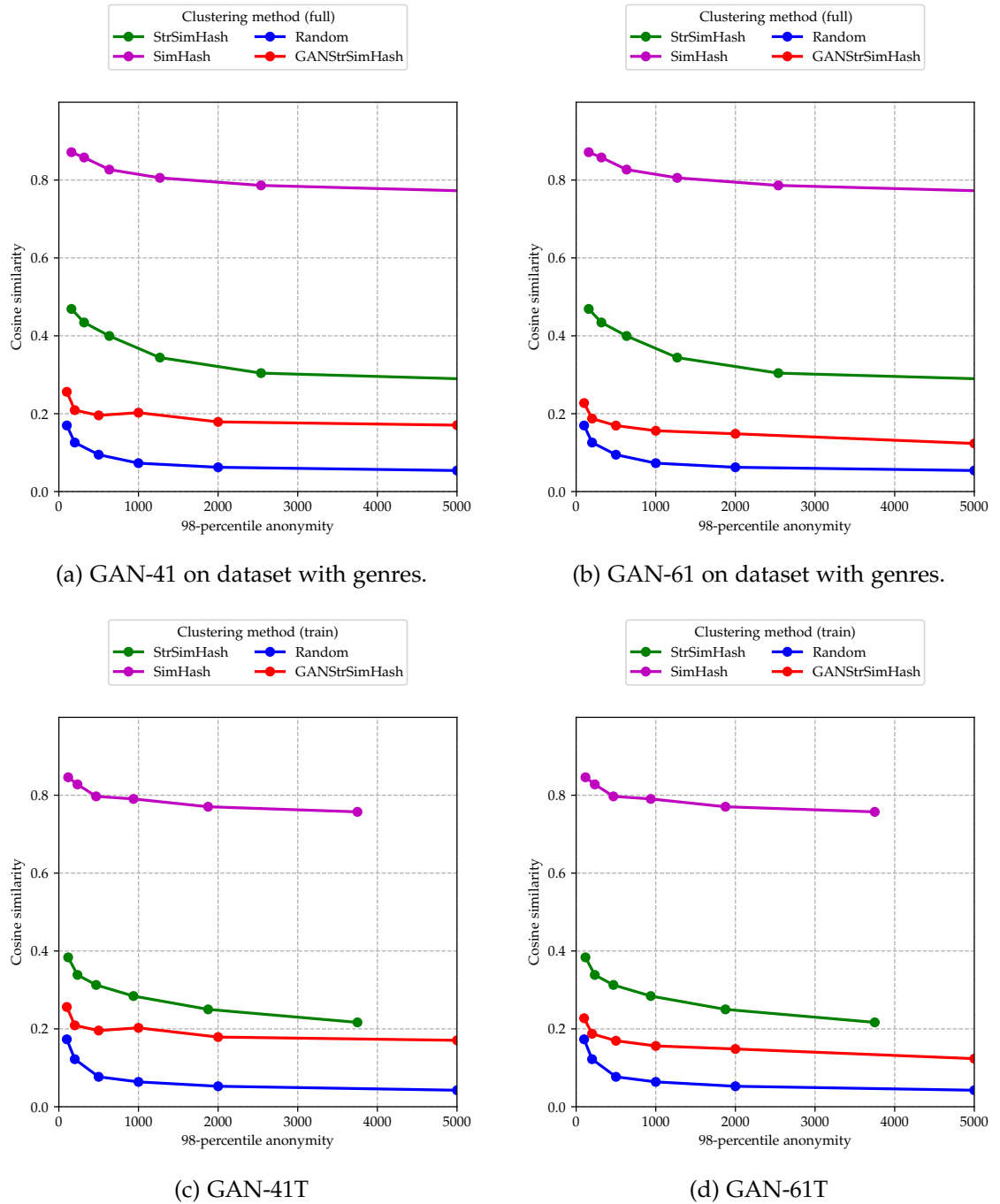
(c) GAN-41T

(d) GAN-61T

Figure 4.4: FLoC whitepaper evaluation with centering.

Table 4.6: Pairwise Cosine Similarity Matrix (GAN VS REAL)

Checkpoint	Common Movies		Common Genres	
	Max Entry	Min Entry	Max Entry	Min Entry
GAN-11	4.723 ± 4.637	0.004 ± 0.062	13.426 ± 2.223	1.145 ± 1.386
GAN-21	4.898 ± 4.487	0 ± 0	13.461 ± 1.843	1.016 ± 1.259
GAN-31	4.355 ± 4.883	0 ± 0	13.520 ± 1.796	0.875 ± 1.111
GAN-41	$\mathbf{4.984 \pm 4.538}$	0 ± 0	13.453 ± 2.005	1.145 ± 1.473
GAN-51	4.277 ± 5.477	0.004 ± 0.062	13.238 ± 1.980	1.121 ± 1.405
GAN-61	4.051 ± 4.466	0 ± 0	$\mathbf{13.605 \pm 1.834}$	0.855 ± 1.030

Having one such matrix for each cluster for the SimHash bit range going from 5 to 10, it seems reasonable to limit ourselves to those values. We will also fix a SimHash prefix length.

We expect metric being interdependent. For example, take the number of common movies. We expect two vectors with high cosine similarity to be very similar and hence have a higher number of common movies than two vectors with low cosine similarity.

However, if we center the features before computing the cosine similarity, then the features would not be nonnegative anymore. We can now have negative cosine similarities. And the interpretation we make of it changes, as already mentioned in Appendix D.4.1. Namely, centering can alter the direction and angle of the vectors, which the cosine similarity rely on.

Without centering, the lowest cosine similarity would be equal or close to zero. And it would likely be two feature vectors that do not share any genres together. Nonetheless, when applying centering, neither the lowest cosine similarity nor the closest to 0 would have zero genres in common. Also the opposite could happen, movie histories are deemed very similar when not sharing common movies, and the genres distribution is somewhat different.

Results

In Table 4.6, we compare the pairwise cosine similarity matrices' extrema entries. To obtain the averages and standard deviations we fixed the SimHash length to 8 bits and accumulated the entries for each SimHash values (256 in total). In order to ease the interpretation of the results, we do not apply centering.

We added the common genre metric as in the FLoC whitepaper evaluation method from Section 4.2.1 with genres used for features extraction. Hence the cosine similarities are computed on the representation of those genres.

For the max entries we expect to see more common movies/genres, but for min entries we expect fewer of them in common. Since we are dealing with
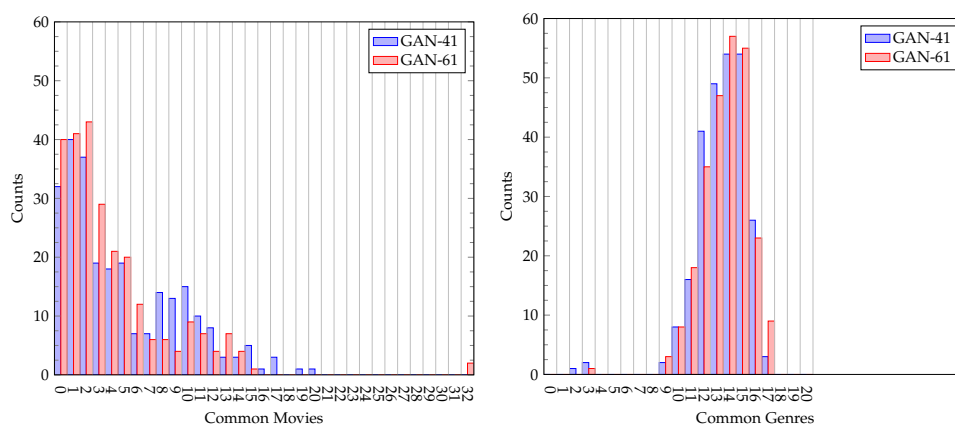
Figure 4.5: Histogram of the Distribution of Common Movie/Genre Counts

the pairwise cosine similarities of users (real and generated) in the same cluster, we would also want the min entries to be as high as possible. Hence for the max and min entry columns we define higher as better.

In Table 4.6 we see that GAN-41 has the highest average number of common movies for the max entries. However, GAN-61 has a higher average number of common genres. As already said earlier in Appendix D.4.1, our metrics try to summarize complex situations into a number. Hence there is not a single best metric. One has to find the best trade off between the metrics of interests. Unfortunately the common movies metric is not very robust since it has very high variance.

In Fig. 4.5, we report the distributions of the common movies and genres metrics for LeakGAN at two different checkpoints. The ones that performed well in Table 4.6.

For the metrics we report, it is important to recall that the movie histories generated by LeakGAN will contain less movies than the training data. Since the integer program takes a subset of the history that matches the target SimHash.

It can happen that the GAN generates the same movie history as in the training data. GAN-61 is able to generate histories that, apart from the order, exactly match the training data. We observed across different runs that GAN-61 generated histories with above 25 common movies. However, it still remains that GAN-41 is more consistent at generating higher average common movie counts. For the common genre the distance between the two distribution is not significant.

We observed that GAN-41 was generally the best performing across most of the metrics. However, due to the high variance of certain metric (e.g., common movies) the improvements are not noticeably significant. Whereas

for other metric, like the minimum hamming distance, GAN-41 was a clear winner. With this metrics, we were able to evaluate the quality of the GAN's history generation for different model from the same training loop. These metrics can thus be used during training either individually or together. When evaluating more than one metric the aim is to find a trade-off between them. This evaluation demonstrated that our attack is able to generate plausible histories similar to that of real users mapped to the same cohort.

Chapter 5

Related Work

5.1 Criticism and Analysis of FLoC

Google's Federated Learning of Cohort is a tool part of their Privacy Sandbox project aimed at "building a more private [and] open web" [38]. After Google's announcement of those plans a lot of criticism emerged. We summarize the criticisms brought by NGOs (e.g., EFF [17]), issues posted on FLoC Github [64], browser competitors (Mozilla [52], Brave [56], and Vivaldi [59]).

Fingerprinting   Browser fingerprinting consists in collecting persistent information from users that uniquely identifies their browser [22]. Examples of the collected information are user agent, font rendering capabilities, or individual choices as Do-Not-Track. Chrome is behind competing browsers like Safari or Firefox in the fight against fingerprinting. In addition, FLoC increases the fingerprinting surface, since the tracker only has to identify a user among a group of thousands of users while before it was among millions of users. Google promised to address fingerprinting with its "Privacy Budget" proposal [35] currently planned for 2023 [38]. However, rival browsers like Brave and Mozilla are doubtful about the success of such "budget" techniques [51, 57]. They conclude that FLoC constitutes a new fingerprinting source, in a context where Google is not able to deal properly with existing ones, and therefore it seems counter productive to deploy FLoC in the current situation.

Use of external information   FLoC IDs seem to be able to do the job that trackers did with third-party cookies, except FLoC is doing it for them. The trackers could also add their own information. Now they only have to differentiate among people in the cohort instead of the whole population. In addition, sequence of cohort ID could be tracked and used to uniquely identify users, as mentioned in issue #100 [64] . FLoC advertises with its

ID a summary of one user's current history to every websites. This can have negative effects, websites could learn information they should not. For example, if one visits an insurance website, they could refuse coverage if they found out (e.g., with the help of the FLoC ID) about some underlying condition that would diminish their profits. Furthermore, it might be possible for tracker to reverse the FLoC ID and get a good idea of what websites a user has in its history. Resulting in potentially sensitive information about some specific demographic leaking to trackers. Chrome used the same sensitive categories for FLoC as the one used for Google targeted ads. However, sensitive information can still leak in forms that are less apparent. A well-known example[1] is an angered father finding Target is sending baby clothes advertisement to his teenage girl. Target knew about his daughter pregnancy before him through targeted advertisement.

Unfairness potential The EFF says "The power to target is the power to discriminate". Such advertising techniques enable one to target specific groups while discarding others. This can be discriminatory when used to restrict access to loan or job listings. Google solution seems to mostly rely on filtering out sensitive cohorts. Nevertheless this is generally not sufficient. This filtering also raises concerns about censorship or discrimination. In addition, malicious advertisers would benefit from plausible deniability. Indeed, they do not target explicitly people in sensitive categories. Some also reported the use of this sensitive information (e.g., religion), that could be inferred by dictatorship on the visit of national websites, to discriminate a part of its population. In addition, FLoC is more favorable to big companies (like Google or trackers company). Those companies would be able to infer a lot more information from the cohort ID, due to the huge amount of traffic they can observe on the different websites they use. Therefore, it will be harder for smaller companies with less observations to extract meaning form statistical analysis. Another harmful example reported by Brave is audience stealing. A website selling niche products might loose customers if FLoC advertise to other parties their customers' interests. Rival browsers reported that FLoC implementation might hinder their own existing mitigations to protect users. Therefore, there does not seem to be enough countermeasures to prevent FLoC being used for exploitation, discrimination or harm.

Lack of transparency There has been a lot of criticism on the lack of transparency and auditing in FLoC development. Some targeted the black-box nature of certain operation. For example the server-side cohort ID assignment, since data is being sent to Chrome servers to ensure *k*-anonymity and remove sensitive cohort. That server needs to be trusted and auditable.

---

[1] https://www.businessinsider.com/the%2Dincredible%2Dstory%2Dof%2Dhow%2Dtarget%2Dexposed%2Da%2Dteen%2Dgirls%2Dpregnancy%2D2012%2D2

For now, Google only published some detail explaining the procedure and needs to be trusted without external validation. Others advocated for more meaningful cohort names. They wanted a meaning to already be given to the cohort ID, as opposed to the current numbers used to define cohort where the meaning has to be extracted from tracking.

Debatable privacy improvements   Some posted issues and browser vendors denounced the false or misleading claims about FLoC advertised privacy properties. For example, k-anonymity protections prevents from distinguishing a user in a group. However, one can still learn private and valuable information that one might not want to be shared without explicit approval. Furthermore, some criticize Google statements. In particular Google says FLoC is an improvement over third-party cookies, which is a terrible baseline to take. They also say FLoC improves privacy, while mostly comparing it to Chrome's default and not the other more privacy-preserving browser. Brave also adds that filtering sensitive cohort generally necessitate the gathering of sensitive information. Moreover, what is sensitive might be up to everyone individually and not to Google's definition. In summary, EFF and competing vendors do not see real privacy improvements with this Google proposal and judge the countermeasures to the above problems insufficient.

5.2   Alternatives to FLoC

5.2.1   Contextual Advertisement

Contextual information can be obtained without having to target a specific user based on personal information. It can look for specific keywords on a website and show ads based on them. This is less privacy-infringing, was the standard in the past, and does not require developing multiple new tracking protocol allegedly more respectful of your privacy. This is what EFF [17] advocates, along with the conclusion of [7]. Approaches premised on tracking users would need to prove that they indeed protect user privacy. Berke et al. [7] showed that FLoC failed in that regard.

5.2.2   Targeted Advertisement

If contextual advertising is not sufficient, then others proposed targeted advertising while being more respectful of users' privacy.

A blog post[2] from Mozilla already lists a few existing proposals for more privacy-preserving advertising. It mention what other browser manufacturers developed like Google's privacy sandbox [38]. Microsoft PARAKEET [41] proposes to modify ad requests such that the browser anonymizes personal

---

[2]https://blog.mozilla.org/en/mozilla/the-future-of-ads-and-privacy/

identifiable information, but the rest of the ecosystem does not change once it has the anonymized information. Adnostics [60] proposal lets the user's browser choose the targeted advert. It makes billing more challenging as the ad-network does not know which ads was shown. Therefore, they also change the billing system using homomorphic encryption. "The ad network remains agnostic to the user's interests"[3]. One can try Adnostic using a Firefox extension. There is also a variant of Adnostic called AdScale that improve scalability and can support billions of daily ad impressions [26]. There also exists other browser extensions like Crumbs[4] which offer to protect your privacy by anonymizing your personal information. It then shares interests, based on your personal data but that should not allow identification. This is in a sense what FLoC successor's the Topics API would do. Nevertheless, it is important to remember that anonymized data can still allow identification of users when not done properly [43].

We have only discussed some of the existing proposals for privacy-preserving targeted advertisements. We also found some related works on building privacy-preserving recommender systems [62]. It is in a sense related to targeted advertisements. For example one could see movie recommendations as a recommendation for items to buy on a websites.

There are multiple proposals for a more privacy-preserving advertisement. Nevertheless, there is still no consensus on the best method to achieve it. And some of the proposed methods can be hard to analyze.

5.2.3 Federated Advertisement

FLoC initially planned to incorporate Federated Learning in its protocol. [34] saw the idea of using federating learning as promising: "[it] offer[s] all the benefits of machine learning without the drawbacks of centralized data collections". Noting that Google already uses it. For example Gboard (a keyboard app) uses it to provide customized word predictions. However, the division of users in cohort seemed unnecessary.

There does not seem to be a lot of applications of federated learning for targeted advertisements. However, there is research [42], which Brave plans to use to recommend news to users without collecting profiles on them. If federated learning can be used successfully to recommend news to users, it should also be able to recommend ads to show.

---

[3]https://crypto.stanford.edu/adnostic/
[4]https://crumbs.org/

5.3   Attacks on Targeted Advertising

For most of the points in 5.1, critics provided potential attacks. Sometimes the Google team also proposed ideas to mitigate the issues.

Considering that FLoC only ran in an origin trial with a small percentage of users, the majority of the attacks remained theoretical rather than real implementations. An exception is [7], which implemented and analyzed their attacks. They used proprietary (paid) demographic and browsing history data instead of movie dataset like we did. Then they emulated the cohorts FLoC would produce for their analysis. One attack idea is to track sequence of FLoC IDs. This attack was first proposed in issue #100 of FLoC repository [64], but without practical implementation. However, in [7] the attack was realized. By only tracking the sequence of FLoC IDs over time, they were able to uniquely identify 95% of user's devices after 4 weeks. This attack would be even more efficient using standard fingerprinting techniques. Therefore, the stated goal of FLoC preventing individualized user tracking is not achieved. In addition, with the observed data, they could connect user racial backgrounds to their browsing histories, but they found no direct connection between race and cohorts.

Other issues suggested methods to recover the browsing history. For example in issue #40, they suggest to use the popularity of websites (e.g., trending articles in a given time frame) and known website that are recurrently used (e.g., news site, social media, bank account). As user's browsing history are not random, an attacker could precompute possible FLoC IDs and gain valuable insights on the sites a user could have visited.

This is in the spirit of our own attack, except that we do not use a priori information about our end users with our GAN. But we may incorporate to some extent the popularity of websites in our training data. On the other hand, we use more advanced techniques than only SimHash matching to perform the attack, and we also implemented a proof of concept.

Furthermore, as a minimum cohort size is required for k-anonymity protection, one can mount a Sybil attack. This was already reported by Mozilla [52]. Our attack provides a practical implementation using the SimHash preimage attack or integer programming. With this attack one can generate enough users so that a particular cohort is saturated. This would force Chrome server-side pipeline to use a greater prefix length for the SimHashes mapped to the target cohort. With a longer prefix the cohort should contain users sharing more similar browsing patterns. And as we populated this cohort with generated users they should be fewer real users in the cohort, which then may be easier to identify.

Other issues went even beyond the previous attack, to consider what harm it could enable in society. Issue #36 takes as a real world example the use

of dating apps to target and abuse LGBT people[5]. It raised concerns on FLoC allowing malicious individuals to target members of a cohort sharing a specific trait. Since it should be easy to emulate the browsing history of members of this group who share known common interests. And even in the case where Chrome removal of sensitive groups would work. It could discriminate markets with genuine activities who wish to target such groups.

Additionally, issue #38 describes that malicious websites could force users into arbitrary cohort, by embedding code to open websites in the background. A proposed mitigation was to only include history created due to user input. This seems to highlight a Google's deploy-first mitigate-later politic.

5.4 Usage of GANs for Hash Reversal

5.4.1 Perceptual Hash Reversal

A perceptual hash is a class of locality-sensitive hash (LSH). As we saw in Section 2.2.1, SimHash is also part of the LSH family.

GANs have been used to reverse perceptual hashes, for example by Locascio [39]. It trains Pix2Pix [29] on a dataset. Pix2Pix learns a mapping from the input images to the output images. To transform the perceptual image hash into something Pix2Pix understands it is arranged into a two dimensional array (interpreted as an image). The image hash is used as input for Pix2Pix which then generates a new image similar to the one used to compute the input image hash. However, the output image may not have exactly the same hash as the original. To increase the probability of generating a hash collision, one can modify the GAN's objective to improve reversibility. Adding a term to the Pix2Pix loss, measuring the distance between the generated and original hash, can help guide the training. From his experiments, this modification improved the generated image's hash collision rate from approx. 30% to 80%.

For our attack, the GAN objective was to generate text and not learn a mapping from text to text. Therefore, we did not follow the same approach. In the case where we had a similar GAN for text, it is also not obvious how to pipe the SimHash as input text to our GAN. Generally for text, if we use a vocabulary it is not obvious how to transform the SimHash into the vocabulary.

This example suggests not using perceptual hashing, at least without supplementary countermeasures, for application where protecting privacy must prevent leakage of the preimage.

---

[5]https://www.pinknews.co.uk/2015/02/18/gay%2Ddating%2Dapps%2Dused%2Dby%2Dattackers%2Dto%2Dtrap%2Dvictims%2Din%2Direland/

5.4.2 Neural Hash Reversal

The Apple Child Sexual Abuse Material (CSAM) Detection [3] intends to store hashes of illegal images. However, these hashes should be somewhat robust to various image transformations (e.g., crop, rotation etc.). Therefore, Apple trained a CNN to detect slightly transformed image so that they can be matched to the same hash.

Nevertheless, such neural networks are vulnerable to adversarial attacks [24], adding specially crafted noise (perturbations) to an image changes its classification. These attacks can be made more efficient using GANs to generate adversarial samples [68].

Another attacks managed to extract Apple's NeuralHash model[6]. These models then lead to further successful attacks [33, 4].

In our case, FLoC did not make use of a neural hash. Namely, there was no neural network used in the FLoC computation. It is therefore not possible to extract the network model and then generate adversarial examples for it.

---

[6]https://www.reddit.com/r/MachineLearning/comments/p6hsoh/p_appleneuralhash2onnx_reverseengineered_apple/

Chapter 6

Conclusion and Discussion

In this work, we provide an attack on the FLoC implementation's claimed *k*-anonymity protection. We first demonstrated that it is relatively simple to find preimages for a Locality Sensitive Hashing algorithm like SimHash. We then proceed to use this preimage attack with a GAN that generates seemingly realistic user's histories. We can now generate artificial user history profiles whose SimHashes are chosen to target a specific group of users. As we generate more profiles, the cohort they are assigned to gets more specific until it includes only a few real users, which compromises its *k*-anonymity protection. It follows that FLoC allows to distinguish and track users inside the same cohort, which is against its objectives. Therefore, the Google FLoC proposal is not meeting its promises of replacing third-party cookies while preserving privacy of users.

We evaluated the quality of our user history generation. We quantified the improvements made by the GAN's history generation compared to a random history generation (to the best of our knowledge there is no other baseline). Our metrics report substantial improvement over the baseline. However, it is still possible to make further enhancements, since the field of GANs for discrete data is still an active area of research and more suitable models can be developed with their own metrics. Note that such advancements together with simply using orders of magnitude more computational power are realistic, since the attacker model includes Google or another large advertiser to run the attack.

We suggest two countermeasures to our attack. Making the server that assigns SimHashes to cohort ID trusted and unable to read sensitive data would make our attack far more complicated, but not impossible since we can approximate user SimHash by the Sybil attack. The proposal can also leverage differential privacy methods (e.g., to not always return the same cohort and make it vary per website), which would also reduce fingerprintability and provide plausible deniability to users.

After receiving detailed feedback on their FLoC proposal, Google proposed partial mitigations on the possible exploits and changed their view on the privacy guarantees. But since that still did not address all issues, in early 2022 Google announced they would replace FLoC with the Topics API. Similarly to FLoC, the new Topics API still leverages a user's browsing history to compute aggregated information directly in browser that is then shared with advertisers. However, it does not return a cohort ID but a set of topics and the returned topics may differ across sites and time so as to limit the amount of information leakage. This directly prevents the Sybil attack. According to the differential privacy, Topics API outputs a random topic with a 5% probability. The browser will determine the topics from the history of visited websites using a classifier model that is still to be specified. Our attack greatly utilized the rich information in SimHash, while the Topics API will send from the browser only three topics that hold significantly less information. We therefore assume our attack is no longer valid on the Topics API, but since the implementation details have yet to be provided, it is possible that some part of our attack may still apply. This new proposal by Google is an improvement from the previous iteration. However, it remains still to be proven that it achieves its privacy goals as Google still prioritizes utility over privacy.

Bibliography

[1]   Adalytics, ed. *Who is sharing data with Google's FLoC ad algorithm?* 2021.
       URL: https://adalytics.io/blog/google-chrome-floc (visited on
       02/22/2022).

[2]   Josh Karlin Andrés Muñoz Medina Michael Kleber and Marshall Vale.
       Measuring Sensitivity of Cohorts Generated by the FLoC API. 2021. URL:
       https://docs.google.com/a/chromium.org/viewer?a=v&pid=
       sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDo1Mzg4MjYzOWI2MzU2NDgw
       (visited on 02/14/2022).

[3]   Apple. *CSAM Detection Technical Summary*. 2021. URL: https://www.
       apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.
       pdf (visited on 02/17/2022).

[4]   Anish Athalye. *NeuralHash Collider*. 2021. URL: https://github.com/
       anishathalye/neural-hash-collider (visited on 02/17/2022).

[5]   Muhammad Ahmad Bashir and Christo Wilson. "Diffusion of user
       tracking data in the online advertising ecosystem". In: *Proceedings on
       Privacy Enhancing Technologies* 2018.4 (2018), pp. 85–103.

[6]   BBC, ed. *France fines Google and Facebook over cookies*. 2022. URL: https:
       //www.bbc.com/news/technology-59909647 (visited on 02/22/2022).

[7]   Alex Berke and Dan Calacci. *Privacy Limitations Of Interest-based Adver-
       tising On The Web: A Post-mortem Empirical Analysis Of Google's FLoC*.
       2022. arXiv: 2201.13402 [cs.CY].

[8]   Thierry Bertin-Mahieux et al. "The Million Song Dataset". In: *Proceed-
       ings of the 12th International Conference on Music Information Retrieval
       (ISMIR 2011)*. 2011.

[9]   Alexander Bleier. "On the Viability of Contextual Advertising as a
       Privacy-Preserving Alternative to Behavioral Advertising on the Web".
       In: *Available at SSRN 3980001* (2021).

[10] Andrew Brock, Jeff Donahue, and Karen Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2019. arXiv: 1809. 11096 [cs.LG].

[11] Matt Burgess. *Europe's Move Against Google Analytics Is Just the Beginning*. Ed. by Wired. 2022. URL: https://www.wired.com/story/google-analytics-europe-austria-privacy-shield/ (visited on 02/22/2022).

[12] Liqun Chen et al. *Adversarial Text Generation via Feature-Mover's Distance*. 2020. arXiv: 1809.06297 [cs.CL].

[13] Xinlei Chen et al. *Microsoft COCO Captions: Data Collection and Evaluation Server*. 2015. arXiv: 1504.00325 [cs.CV].

[14] Josh Karlin @ Google Chrome. *Let's Talk About FLoC at PEARG - IETF 111*. 2021. URL: https://datatracker.ietf.org/meeting/111/materials/slides-111-pearg-lets-talk-about-floc-00 (visited on 01/17/2022).

[15] CNIL, ed. *Use of Google Analytics and data transfers to the United States: the CNIL orders a website manager/operator to comply*. 2022. URL: https://www.cnil.fr/en/use-google-analytics-and-data-transfers-united-states-cnil-orders-website-manageroperator-comply (visited on 02/22/2022).

[16] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009. Chap. 16.

[17] Bennett Cyphers. *Google's FLoC Is a Terrible Idea*. Ed. by Electronic Frontier Foundation. 2021. URL: https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea (visited on 02/15/2022).

[18] Li Deng. "The mnist database of handwritten digit images for machine learning research [best of the web]". In: *IEEE signal processing magazine* 29.6 (2012), pp. 141–142.

[19] CVE Details. *Google Cityhash Security Vulnerabilities*. 2012. URL: https://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-23646/Google-Cityhash.html (visited on 01/26/2022).

[20] Shizhe Diao et al. "TILGAN: Transformer-based Implicit Latent GAN for Diverse and Coherent Text Generation". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4844–4858. DOI: 10.18653/v1/2021.findings-acl.428. URL: https://aclanthology.org/2021.findings-acl.428.

[21] John R Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[22] Peter Eckersley. "How unique is your web browser?" In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2010, pp. 1–18.

[23] Alessandro Epasto et al. "Clustering for Private Interest-based Advertising". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2802–2810.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[25] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[26] Matthew Green, Watson Ladd, and Ian Miers. "A protocol for privately reporting ad impressions at scale". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1591–1601.

[27] Jiaxian Guo et al. *Long Text Generation via Adversarial Training with Leaked Information*. 2017. arXiv: 1709.08624 [cs.CL].

[28] F Maxwell Harper and Joseph A Konstan. "The MovieLens Datasets: History and Context". In: *ACM Transactions on Interactive Intelligent Systems* 5.4 (Dec. 2015), 19:1–19:19. ISSN: 2160-6455. DOI: 10.1145/2827872. URL: http://doi.acm.org/10.1145/2827872.

[29] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[30] Josh Karlin. *Topics API GitHub*. 2022. URL: https://github.com/jkarlin/topics (visited on 02/19/2022).

[31] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[32] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[33] Lim Swee Kiat. *Apple NeuralHash Attack*. 2021. URL: https://github.com/greentfrapp/apple-neuralhash-attack (visited on 02/17/2022).

[34] Marc Langheinrich. "To FLoC or Not?" In: *IEEE Pervasive Computing* 20.2 (2021), pp. 4–6. DOI: 10.1109/MPRV.2021.3076812.

[35] Brad Lassey. *Privacy Budget GitHub*. 2021. URL: https://github.com/bslassey/privacy-budget (visited on 02/16/2022).

[36] Victor Le Pochat et al. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. Feb. 2019. DOI: 10.14722/ndss.2019.23386.

[37]   Kevin Lin et al. *Adversarial Ranking for Language Generation*. 2018. arXiv: `1705.11001 [cs.CL]`.

[38]   Google LLC. *Privacy Sandbox*. 2019. URL: `https://privacysandbox.com/` (visited on 01/17/2022).

[39]   Nick Locascio. *Black-Box Attacks on Perceptual Image Hashes with GANs*. Ed. by Towards Data Science. 2018. URL: `https://towardsdatascience.com/black-box-attacks-on-perceptual-image-hashes-with-gans-cc1be11f277` (visited on 02/17/2022).

[40]   Don Marti. *Early Status of the FLoC Origin Trials*. Ed. by CafeMedia. 2021. URL: `https://cafemedia.com/early-status-of-the-floc-origin-trials/` (visited on 02/18/2022).

[41]   Microsoft. *PARAKEET GitHub*. 2021. URL: `https://github.com/microsoft/PARAKEET` (visited on 02/17/2022).

[42]   Lorenzo Minto et al. *Stronger Privacy for Federated Collaborative Filtering with Implicit Feedback*. 2021. arXiv: `2105.03941 [cs.LG]`.

[43]   Arvind Narayanan and Vitaly Shmatikov. *How To Break Anonymity of the Netflix Prize Dataset*. 2007. arXiv: `cs/0610105 [cs.CR]`.

[44]   Sophie J. Nightingale and Hany Farid. "AI-synthesized faces are indistinguishable from real faces and more trustworthy". In: *Proceedings of the National Academy of Sciences* 119.8 (2022). ISSN: 0027-8424. DOI: `10.1073/pnas.2120481119`. eprint: `https://www.pnas.org/content/119/8/e2120481119.full.pdf`. URL: `https://www.pnas.org/content/119/8/e2120481119`.

[45]   Shigeki Ohtsu. *FLoC Simulator in Go*. 2021. URL: `https://github.com/shigeki/floc_simulator` (visited on 02/09/2022).

[46]   Lukasz Olejnik, Claude Castelluccia, and Artur Janc. "Why johnny can't browse in peace: On the uniqueness of web browsing history patterns". In: *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2012)*. 2012.

[47]   Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: `10.3115/1073083.1073135`. URL: `https://aclanthology.org/P02-1040`.

[48]   Geoff Pike and Google Software Engineering Team Jyrki Alakuijala. *Introducing CityHash*. 2011. URL: `https://opensource.googleblog.com/2011/04/introducing-cityhash.html` (visited on 01/26/2022).

[49] The Chromium Projects. *FLoC Origin Trial & Clustering*. 2021. URL: https://www.chromium.org/Home/chromium-privacy/privacy-sandbox/floc (visited on 01/17/2022).

[50] Deepak Ravichandran and Sergei Vassilvitskii. *Evaluation of Cohort Algorithms for the FLoC API*. 2020. URL: https://github.com/google/ads-privacy/blob/master/proposals/FLoC/FLOC-Whitepaper-Google.pdf (visited on 02/09/2022).

[51] Eric Rescorla. "Technical Comments on Privacy Budget". In: (2021). URL: https://mozilla.github.io/ppa-docs/privacy-budget.pdf (visited on 02/24/2022).

[52] Eric Rescorla and Martin Thomson. *Technical Comments on FLoC Privacy*. 2021. URL: https://mozilla.github.io/ppa-docs/floc_report.pdf (visited on 02/15/2022).

[53] Antoine Rouzaud. *FLoC Origin Trial Observations*. Ed. by Criteo. 2021. URL: https://medium.com/@antoine.rouzaud (visited on 02/21/2022).

[54] Allison Schiff. *The Industry Reacts To Google's Bold Claim That FLoCs Are 95% As Effective As Cookies*. Ed. by AdExchanger. 2021. URL: https://www.adexchanger.com/online-advertising/the-industry-reacts-to-googles-bold-claim-that-flocs-are-95-as-effective-as-cookies/ (visited on 02/22/2022).

[55] Prashant Sinha. *Google CityHash in Python*. 2014. URL: https://github.com/prashnts/Hashes (visited on 02/26/2022).

[56] Peter Snyder and Brendan Eich. *Why Brave Disables FLoC*. Ed. by Brave. 2021. URL: https://brave.com/why-brave-disables-floc/ (visited on 02/15/2022).

[57] Peter Snyder and Dr. Ben Livshits. *Brave, Fingerprinting, and Privacy Budgets*. Ed. by Brave. 2019. URL: https://brave.com/web-standards-at-brave/2-privacy-budgets/ (visited on 02/24/2022).

[58] Statista. *User population of selected internet browsers worldwide from 2014 to 2021 (in millions)*. 2022. URL: https://www.statista.com/statistics/543218/worldwide-internet-users-by-browser/ (visited on 01/26/2022).

[59] Jon von Tetzchner. *No, Google! Vivaldi users will not get FLoC'ed*. Ed. by Vivaldi. 2021. URL: https://vivaldi.com/blog/no-google-vivaldi-users-will-not-get-floced/ (visited on 02/15/2022).

[60] Vincent Toubiana et al. "Adnostic: Privacy preserving targeted advertising". In: *Proceedings Network and Distributed System Symposium*. 2010.

[61] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[62] Cong Wang et al. "Toward privacy-preserving personalized recommendation services". In: *Engineering* 4.1 (2018), pp. 21–28.

[63] Mark Weiss. *Digiday Research: Most publishers don't benefit from behavioral ad targeting*. Ed. by Digiday. 2022. URL: https://digiday.com/media/digiday-research-most-publishers-dont-benefit-from-behavioral-ad-targeting/ (visited on 02/22/2022).

[64] WICG. *FLoC GitHub*. 2021. URL: https://github.com/WICG/floc (visited on 01/17/2022).

[65] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[66] Michal Wlosik. *How Different Browsers Handle First-Party and Third-Party Cookies*. Ed. by Clearcode. 2019. URL: https://clearcode.cc/blog/browsers-first-third-party-cookies/ (visited on 02/25/2022).

[67] Worldometers, ed. *World Population*. 2022. URL: https://www.worldometers.info/world-population/ (visited on 02/26/2022).

[68] Chaowei Xiao et al. *Generating Adversarial Examples with Adversarial Networks*. 2019. arXiv: 1801.02610 [cs.CR].

[69] Lantao Yu et al. *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*. 2017. arXiv: 1609.05473 [cs.LG].

Appendix A

Materials

The code for the attack components and evaluation will be available on the following GitHub repository:

- https://github.com/fturati/attack-on-floc

Appendix B

Datasets

FLoC computation uses domain names from a user browsing history. We use Tranco list, which is a research-oriented dataset of the most popular domains. Nevertheless, for privacy reasons, we did not find a suitable public dataset linking browsing history of users and domain names. We can however find other form of user history, for example a user list of watched movies or played songs. Those user histories can then be used to train a GAN.

B.1 Tranco

Tranco [36] contains a ranking of the most popular websites, suitable for research, since it provides better reproduceability than other existing rankings.

For our project we used the Tranco list[1] of the Top 1 million domain names generated on 03 October 2021, nevertheless another list could be used without any issue.

As an example for this dataset we display the top 5 from the previously mentioned list:

1. google.com

2. youtube.com

3. facebook.com

4. netflix.com

5. microsoft.com

We cannot use this dataset for training a GAN to generate plausible user history. Since it contains only the most popular websites and not user browsing histories. Therefore, we only used this dataset with our first preimage

---

[1]Available at https://tranco-list.eu/list/NLKW/1000000

attack (see Section 3.2), since this attack returns preimages regardless of their plausibility.

B.2   Movielens 25m

The MovieLens 25M dataset [28] is a movie ratings dataset. It contains a bit more than 25 millions ratings (from 0 to 5 with 0.5 increment) from 162 541 users across 62 423 movies. For each movie a list of genres is also provided from 20 possible distinct categories. The average length of a user movie list is around 153 while the median is 70.

More details on the dataset can be found in the aforementioned article [28] or in the dataset README[2].

This dataset is also used in the Google FLoC whitepaper [50] to evaluate their proposal.

B.3   COCO Captions

The COCO Image Captions dataset [13] contains captions describing images. Five captions written by different people are provided for training and validation images. LeakGAN [27] reports 20 734 words and 417 126 sentences for the version of the dataset they used. The captions are generally short, most sentences are about 10 words.

We did not directly work with this dataset for our attack but we did run LeakGAN and TILGAN [20] on it. It served as a realistic benchmark to evaluate the quality of the generated sentences.

We include some example captions from the dataset:

A person is taking a photo of a cat in a car.

A stuffed animal is laying on the bed by a window.

The top of a kitchen cabinet covered with brass pots and pans

A phone lies on the counter in a modern kitchen.

A man riding a bicycle on a road carrying a surf board.

---

[2]https://files.grouplens.org/datasets/movielens/ml-25m-README.html

Example Outputs from GAN

In Table 3.1, we presented an output of the attack that is a well matching history. For the sake of fairness, in Table C.1, we provide one "bad" example. The GAN parameters were the same as in Table 3.1, namely a vocabulary size of 5847 and a maximum sequence length of 77.

Toy Story is the first movie in the majority of the histories. We noticed a similar pattern when training the GAN with the COCO Captions dataset, namely that most sentences would start with the letter 'A'. This may be an effect of mode collapse where the GAN would lack in diversity of outputs. It may also be worsen by the training data which has user movie history in increasing number of movie ID. We note that the order of the movie ID is learnt, except for one Harry Potter movie (ID 40815). We also note from the release date of the Harry Potter movie that it is not in accordance with the time span of the release of other movies. We also highlight in blue some movies part of a saga which were not the first release. In another color we show movies that are part of a franchise and were the first of a serie of movies. We observe that this particular history does not contain multiple movies from the same franchise despite having numerous movies part of a franchise.

In Table C.2, we have another randomly choosen example for smaller GAN parameters (vocabulary size of 5002 and maximum sequence length of 32).

For the previous examples the GAN learnt to produce histories with increasing movie IDs, since it was also the case in the training data. We note that changing the preprocessing step by randomly shuffling the movie IDs in the histories (see Section 3.3.2) had an effect on the GAN's diversity in the first generated movie. Indeed, we did not observe anymore the same first generated movie (i.e., Toy Story) in the majority of synthetic histories. We also observe that this change, along with the reduction of the maximum history length to 32, makes it harder for the GAN to generate histories with

Table C.1: "Bad" example output from LeakGAN

TokenID	MovieID	Title	Genres
2298	1	Toy Story (1995)	Adventure, Animation, Children, Comedy, Fantasy
187	2	Jumanji (1995)	Adventure, Children, Fantasy
2713	3	Grumpier Old Men (1995)	Comedy, Romance
1327	10	GoldenEye (1995)	Action, Adventure, Thriller
1267	66	Lawnmower Man 2: Beyond Cyberspace (1996)	Action, Sci-Fi, Thriller
1328	95	Broken Arrow (1996)	Action, Adventure, Thriller
1714	104	Happy Gilmore (1996)	Comedy
988	40815	Harry Potter and the Goblet of Fire (2005)	Adventure, Fantasy, Thriller, IMAX
4216	110	Braveheart (1995)	Action, Drama, War
4418	140	Up Close and Personal (1996)	Drama, Romance
3975	150	Apollo 13 (1995)	Adventure, Drama, IMAX
4713	151	Rob Roy (1995)	Action, Drama, Romance, War
780	160	Congo (1995)	Action, Adventure, Mystery, Sci-Fi
4826	161	Crimson Tide (1995)	Drama, Thriller, War
1431	165	Die Hard: With a Vengeance (1995)	Action, Crime, Thriller
4707	168	First Knight (1995)	Action, Drama, Romance
1432	185	The Net (1995)	Action, Crime, Thriller
536	204	Under Siege 2: Dark Territory (1995)	Action
698	208	Waterworld (1995)	Action, Adventure, Sci-Fi
2978	236	French Kiss (1995)	Action, Comedy, Romance
3315	261	Little Women (1994)	Drama
4634	265	Like Water for Chocolate (Como agua para chocolate) (1992)	Drama, Fantasy, Romance
4628	266	Legends of the Fall (1994)	Drama, Romance, War, Western
5645	281	Nobody's Fool (1994)	Comedy, Drama, Romance
1428	288	Natural Born Killers (1994)	Action, Crime, Thriller
5118	292	Outbreak (1995)	Action, Drama, Sci-Fi, Thriller
5823	296	Pulp Fiction (1994)	Comedy, Crime, Drama, Thriller
5087	315	The Specialist (1994)	Action, Drama, Thriller
697	316	Stargate (1994)	Action, Adventure, Sci-Fi
5529	317	The Santa Clause (1994)	Comedy, Drama, Fantasy
4020	329	Star Trek: Generations (1994)	Adventure, Drama, Sci-Fi
3287	337	What's Eating Gilbert Grape (1993)	Drama
2722	339	While You Were Sleeping (1995)	Comedy, Romance
1666	344	Ace Ventura: Pet Detective (1994)	Comedy
5163	349	Clear and Present Danger (1994)	Action, Crime, Drama, Thriller
5789	356	Forrest Gump (1994)	Comedy, Drama, Romance, War
2708	357	Four Weddings and a Funeral (1994)	Comedy, Romance
2634	367	The Mask (1994)	Action, Comedy, Crime, Fantasy
1510	377	Speed (1994)	Action, Romance, Thriller
3077	380	True Lies (1994)	Action, Adventure, Comedy, Romance, Thriller
2147	410	Addams Family Values (1993)	Children, Comedy, Fantasy
3057	420	Beverly Hills Cop III (1994)	Action, Comedy, Crime, Thriller
2228	432	City Slickers II: The Legend of Curly's Gold (1994)	Adventure, Comedy, Western
1326	434	Cliffhanger (1993)	Action, Adventure, Thriller
2699	440	Dave (1993)	Comedy, Romance
696	442	Demolition Man (1993)	Action, Adventure, Sci-Fi
4738	454	The Firm (1993)	Drama, Thriller
904	457	The Fugitive (1993)	Thriller
1363	480	Jurassic Park (1993)	Action, Adventure, Sci-Fi, Thriller
5308	500	Mrs. Doubtfire (1993)	Comedy, Drama
4383	509	The Piano (1993)	Drama, Romance
3825	527	Schindler's List (1993)	Drama, War
4201	553	Tombstone (1993)	Action, Drama, Western
1142	555	True Romance (1993)	Crime, Thriller
576	589	Terminator 2: Judgment Day (1991)	Action, Sci-Fi
3979	590	Dances with Wolves (1990)	Adventure, Drama, Western
1424	592	Batman (1989)	Action, Crime, Thriller
1187	593	The Silence of the Lambs (1991)	Crime, Horror, Thriller
866	595	Beauty and the Beast (1991)	Animation, Children, Fantasy, Musical, Romance, IMAX
1330	733	The Rock (1996)	Action, Adventure, Thriller
1364	780	Independence Day (a.k.a. ID4) (1996)	Action, Adventure, Sci-Fi, Thriller

68

Table C.2: Example output from LeakGAN with different parameters

TokenID	MovieID	Title	Genres
399	527	Schindler's List (1993)	Drama, War
2919	5989	Catch Me If You Can (2002)	Crime, Drama
3939	59369	Taken (2008)	Action, Crime, Drama, Thriller
3560	36529	Lord of War (2005)	Action, Crime, Drama, Thriller, War
3842	54997	3:10 to Yuma (2007)	Action, Crime, Drama, Western
3999	63082	Slumdog Millionaire (2008)	Crime, Drama, Romance
3803	53121	Shrek the Third (2007)	Adventure, Animation, Children, Comedy, Fantasy
3602	40815	Harry Potter and the Goblet of Fire (2005)	Adventure, Fantasy, Thriller, IMAX
4147	72998	Avatar (2009)	Action, Adventure, Sci-Fi, IMAX
3796	52722	Spider-Man 3 (2007)	Action, Adventure, Sci-Fi, Thriller, IMAX
3460	30707	Million Dollar Baby (2004)	Drama
240	318	The Shawshank Redemption (1994)	Crime, Drama
4059	68157	Inglourious Basterds (2009)	Action, Drama, War
3743	49530	Blood Diamond (2006)	Action, Adventure, Drama, Crime, Thriller, War
3543	34405	Serenity (2005)	Action, Adventure, Sci-Fi
4091	69844	Harry Potter and the Half-Blood Prince (2009)	Adventure, Fantasy, Mystery, Romance, IMAX
1839	3000	Princess Mononoke (Mononoke-hime) (1997)	Action, Adventure, Animation, Drama, Fantasy
793	1240	The Terminator (1984)	Action, Sci-Fi, Thriller
697	1089	Reservoir Dogs (1992)	Crime, Mystery, Thriller
3738	49272	Casino Royale (2006)	Action, Adventure, Thriller
2332	4011	Snatch (2000)	Comedy', Crime, Thriller
2910	5952	The Lord of the Rings: The Two Towers (2002)	Adventure, Fantasy
4654	112552	Whiplash (2014)	Drama
2887	5816	Harry Potter and the Chamber of Secrets (2002)	Adventure, Fantasy
2641	4896	Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)	Adventure, Children, Fantasy
224	296	Pulp Fiction (1994)	Comedy, Crime, Drama, Thriller
4835	148626	The Big Short (2015)	Drama
4306	85414	Source Code (2011)	Action, Drama, Mystery, Sci-Fi, Thriller
3937	59315	Iron Man (2008)	Action, Adventure, Sci-Fi
4772	134130	The Martian (2015)	Adventure, Drama, Sci-Fi
2803	5459	Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002)	Action, Comedy, Sci-Fi
412	541	Blade Runner (1982)	Action, Sci-Fi, Thriller,

multiple movies from the same franchise. Since the histories are truncated and randomly subsample movies from the real histories. The presented history contained four Harry Potter movies among the first six that were released. It is the only franchise that has several movies in the generated history. We also highlight in blue some movies part of a saga which were not the first release. In purple we show some movies that are part of a franchise and were the first of a serie of film.

Appendix D

FLoC Whitepaper Anonymity Evaluation Details

In Section 4.2.1, we presented a method used by Google researchers to evaluate the anonymity of their FLOC proposal. The whitepaper [50] did not contain a thorough explanation of the steps they made to obtain their results but in the following we will detail ours.

D.1 Dataset

We used the MovieLens dataset [28], more detail about this dataset can be found in Appendix B.2. The FLoC whitepaper also made a comparison with the Million Song Dataset [8], which showed similar results. Therefore we omit this dataset.

D.2 Feature Extraction

It is easier to work with numerical values than text. For downstream computation, we need the user data mapped into a vector space as follow:

- For each movie in a user rating list we create a vector with 20 components (one for each possible genre/category). Each element of this vector, let us call them category weight,[1] is taken to be 1 or 0 if the movie is of this genre or not respectively. This category weight is then multiplied by the user rating (from 0 to 5 with 0.5 increment). When this rating is not present (e.g., for the movie list generated by our GAN model) we set it to one for every movie. As 1 is the multiplicative identity, the rating as no effects.

---

[1]The values this weight could take were not defined in the whitepaper so we came up with what seemed to make the most sense

- After this step, to aggregate those vectors we take the average of all the movie features to obtain one feature vector for each user.

- Finally, the features are centered to have mean zero, which means each of the 20 features column has its mean subtracted.

D.3 Cluster Assignment

D.3.1 Random

This is just a random assignment of users to clusters. We fix the size of clusters, and every cluster is of the same size, except one if the number of users was not a multiple of the size.

D.3.2 SimHash

This SimHash is different from the one used in Chromium. The main difference is that the input is the feature vector computed according to Appendix D.2. We refer to FLoC whitepaper for the precise definition.

It did not seem obvious how they used it to perform the cluster assignment. It was not specified how they chose the random unit vector. Therefore, we decided to use a fixed set of random unit-norm vectors for all feature.

This is slightly different from Chromium's SimHash where those random vectors are determined by the SimHash output bit and the input domain hash.

The whitepaper also did not specify how they made the cluster size vary. It makes sense to vary the SimHash bit length. Taking into account the total number of user in the dataset, to obtain different cluster sizes we can change the bit length among values in the 5 to 10 range. However it is not possible to guarantee a minimum number of users per cluster.

D.3.3 Chromium SimHash

Section 3.1.2 details the implementation of SimHash in the Chrome browser. The important part is that this cluster assignment is not using the genre feature extraction discussed previously. It uses the movie titles from the user histories.

To make the cluster sizes change, we made the SimHash bit length vary between 5 and 10 bits. Nonetheless, it was not stated if the whitepaper followed the same approach.

D.3.4 GAN Pipeline Chromium SimHash

In Section 3.5.3, we generated synthetic movie histories. Then we use the integer program to find a subset that matches a target SimHash. Note that for this part we did not apply the discriminator on the history subset returned by the integer program. With this approach, we can easily choose the target SimHash and the number of user we want in each cluster. Since we can generate as many as we see fit, we can ensure a minimum cluster size.

For the cluster assignment we also made the bit length vary between 5 and 10 bits. The corresponding cluster sizes are 5000, 2000, 1000, 500, 200, 100. As the GAN generates histories in batch we might have more users in the clusters than the cluster sizes specify.

D.3.5 Sorting LSH and Affinity Centroid

We do not reproduce the affinity centroid clustering described in [50], since it is centralized and requires sending raw data (and not SimHash of the raw data) to the central server.

Prefix LSH (see Section 3.1.3) is a variant of Sorting LSH. They both serve the same purpose of ensuring cohorts have the required minimum size. Their implementation also requires a central server. Therefore, they were not reproduced either. Note that to compute the Sorting LSH clusters from the SimHash cohorts, one applies the mapping distributed to the browser by the Chrome-operated central server.

D.4 Evaluation

D.4.1 Further Remarks on the Procedure

In FLoC whitepaper [50, Fig. 4], it was not specified if the complete Movie-Lens dataset was used. Specifically, since we see the curves for SimHash and Affinity centroid not having the same data point as for Sorting LSH and Random clustering (see Fig. 4.3). We may wonder if they use some subset of the dataset. For our purposes, to make comparison with the GAN more fair, we also run the experiment on the training data. As it is the only dataset the GAN has seen during training.

However, this leads to some changes in the feature extraction procedure. The whitepaper's feature extraction relied on user's movie ratings (see Appendix D.2). For a browsing history, we can find the topics of a website with a model or by API such as Google's API[2], WebShrinker[3] or SimilarWeb[4].

---

[2]https://cloud.google.com/natural-language/docs/categories
[3]https://www.webshrinker.com/
[4]https://www.similarweb.com/category/

73

For a movie the genres are suitable to perform feature extraction. However, the rating a user gave to a particular movie is more privacy infringing and such a counterpart would not be available for websites. In FLoC there was no mention of the the use of any ratings. Consequently, our GAN does not generate user ratings. In addition, we did not store the user IDs when generating our training data. It is thus not possible to recover the ratings associated to the movie histories. Therefore, the only place ratings can be used is with the full MovieLens dataset. As a reminder from Appendix D.2, when the ratings are missing they are replaced by 1.

In machine learning it is sometimes preferable to have input data with 0 mean. Centering conserves the euclidean distances. However, in the case of FLoC whitepaper's evaluation, centering the data can alter the angles or directions of our vectors. For example, all features are nonnegative, so if we apply centering the small features in the same direction as bigger valued feature vector would get opposite direction after centering. Those vector's properties changed by centering are later used by the cosine similarity.

A reason for using centering, apart from the fact that the FLoC whitepaper does so, is to make use of the full cosine similarity range ($[-1, 1]$). As our input data features otherwise would all be nonnegative and in the $[0, 5]$ range ($[0, 1]$ without ratings). However, as it can be seen in Section 4.2.2, this had effects when we want to extract meaningful results using cosine similarity from the movies dataset. It is hard to reason about a negative cosine similarity when all our feature vector were nonnegative before centering. Centering does alter the interpretation we make of the cosine similarity. In addition, the cosine similarity does not have a notion of distance. It knows the angle between users in the same clusters, but not the distance that separates them.

Our metrics are trying to simplify relation of high-dimensional vectors into a single number. This is not simple, and every metric has its own drawbacks. Nonetheless, we can still extract some value from the whitepaper's metric with centered data.

D.4.2 Results without Centering

As we pointed out in Appendix D.4.1, the whitepaper evaluation method uses centering. However, it alters the interpretation we can make about the cosine similarity. We also reproduced the plots without the centering, see Fig. D.1. The setting is the same as for Fig. 4.4 except for the omitted centering.

As a reminder, the first row (Figs. D.1a and D.1b) is on the full dataset, using ratings except for the GAN. The second row (Figs. D.1c and D.1d) shows the results on the training data subset without ratings.
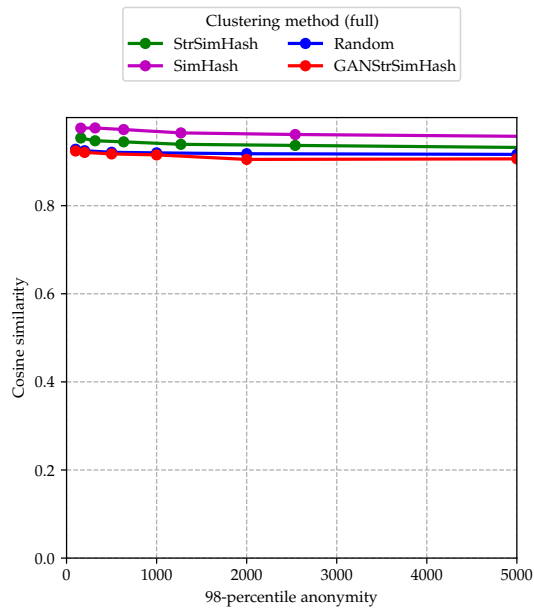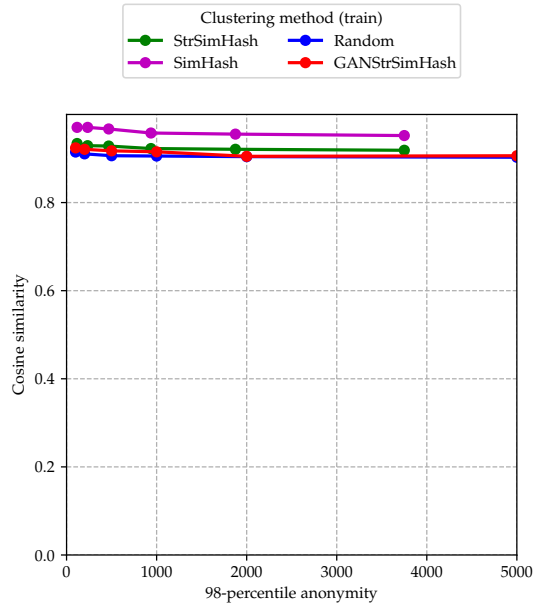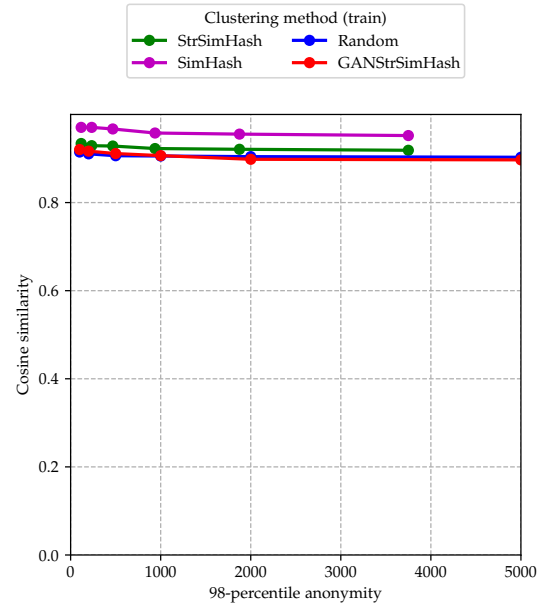
(a) GAN-41 on dataset with genres
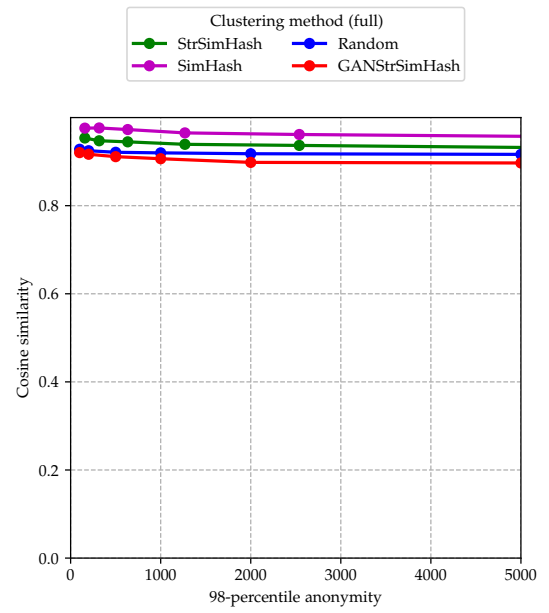
(b) GAN-61 on dataset with genres

(c) GAN-41T

(d) GAN-61T

Figure D.1: FLoC whitepaper evaluation without centering.

Without the centering step our GAN seems to perform worse than random clustering on the full dataset and comparably on the training subset. This is because the other method performs worse on the training dataset. As we already mentioned earlier, this could be due to some particularity of the training dataset. It has shorter histories and does not use ratings (not available for the GAN in all plots). The training data also contains histories that the GAN has seen so it can generate them. The full dataset contains more than 60 000 movies while our GAN can only generate 5 000. In any case, the fact that our GAN performs better on the training dataset is promising. As they are the closest to what the GAN learned. And a better, more powerful, GAN could improve on those results.

GAN-41 performs better than GAN-61. It would need to be consistent across more runs with other metrics (e.g., common movies in Section 4.1.1, minimum hamming distance in Section 4.1.2, etc.) so that we can attribute it to the model performance rather than random fluctuation. From the several runs we made with this metrics, the plot fluctuated and the improvements of GAN-41 were not that significant.

Now we might wonder why every method is performing so well when we only changed the centering step. Without centering all the features are nonnegative ($[0,1]$ range or $[0,5]$ with ratings). Each feature vector has 20 dimensions. In high dimension we have to beware of the curse of dimensionality. The space is so vast that the available data is scarce and scattered. In addition, the feature vector that we have are sparsed, some genres are far less represented than others, meaning some values are close or equal to 0. These reasons can explain the great performance of every clustering method.

The centering step should help regrouping sparse data points together. This could help for later computations. As these same features are used to compute the average cosine similarities reported in the plots.