Privacy Side Channels in Machine Learning Systems

Edoardo Debenedetti, *ETH Zurich;* Giorgio Severi, *Northeastern University;*
Nicholas Carlini, Christopher A. Choquette-Choo, Matthew Jagielski, and
Milad Nasr, *Google DeepMind;* Eric Wallace, *UC Berkeley;* Florian Tramèr, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity24/presentation/debenedetti

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Privacy Side Channels in Machine Learning Systems

Edoardo Debenedetti[1] *Giorgio Severi*[2] *Nicholas Carlini*[3] *Christopher A. Choquette-Choo*[3]
Matthew Jagielski[3] *Milad Nasr*[3] *Eric Wallace*[4] *Florian Tramèr*[1]
[1]ETH Zurich [2]Northeastern University [3]Google DeepMind [4]UC Berkeley

Abstract

Most current approaches for protecting privacy in machine learning (ML) assume that models exist in a vacuum. Yet, in reality, these models are part of larger *systems* that include components for training data filtering, output monitoring, and more. In this work, we introduce *privacy side channels*: attacks that exploit these system-level components to extract private information at far higher rates than is otherwise possible for standalone models. We propose four categories of side channels that span the entire ML lifecycle (training data filtering, input preprocessing, output post-processing, and query filtering) and allow for enhanced membership inference, data extraction, and even novel threats such as extraction of users' test queries. For example, we show that deduplicating training data before applying differentially-private training creates a side-channel that completely invalidates any provable privacy guarantees. We further show that systems which block language models from regenerating training data can be exploited to exfiltrate private keys contained in the training set—even if the model did not memorize these keys. Taken together, our results demonstrate the need for a holistic, end-to-end privacy analysis of machine learning systems.

1 Introduction

In the absence of safeguards, machine learning (ML) models leak private information about their training data [28, 71]. Numerous methods have been proposed to measure and mitigate privacy leakage, including formal techniques [1, 23, 27] and heuristics [42, 47]. However, existing privacy-preserving methods largely assume that ML models *exist in a vacuum*, when in reality ML models are part of larger *systems* that include components for training data filtering, input preprocessing, output monitoring, and more. These system-level components are widely incorporated into real-world ML systems to maximize accuracy, security, and robustness.

In this work, we introduce *privacy side channels*: attacks that exploit system-level components to extract private information at much higher rates than is otherwise possible for isolated ML models. We show that adaptive adversaries of varying strengths—ranging from black-box query access to data poisoning capabilities—can mount privacy attacks that are otherwise impossible without side channels (e.g., revealing test inputs). We evaluate the impact of these side-channel attacks on end-to-end systems, including GitHub Copilot—a black-box production system with millions of users. Concretely, we propose four categories of attacks that span the entire ML lifecycle (overview in Figure 1):

- **Training data filtering (Section 3).** Most large-scale training sets are filtered to remove duplicates and abnormal examples [2, 42, 47]. We demonstrate that data filters introduce side channels because they create dependencies between different users' data. In turn, adversaries can amplify privacy attacks by inserting poison examples that maximize these dependencies. Perhaps most surprisingly, we show that data deduplication [47]—a technique designed to *improve* privacy—can make privacy *worse*, even causing violations of naive differential privacy (DP) guarantees (Section 5). Aside from deduplication, we propose similar attacks for defenses against data poisoning [13, 19, 29].

- **Input preprocessing (Section 4).** Many models require their inputs to be preprocessed, e.g., language models require text to be tokenized. When these preprocessors are built using training statistics (e.g., tokenizers), we show that it creates side channels that allows adversaries to extract private information such as rare training words [66].

- **Model output filtering (Section 4).** To improve privacy, many ML systems include filters that prevent the system from outputting verbatim training data [31, 40]. We show that this creates a side channel that actually *reduces* privacy, even enabling near-perfect membership inference. We use this attack to reverse engineer details of GitHub Copilot's training set (namely, its exact training data cutoff) and then use it to extract OpenSSH private keys from a public language model's training set.

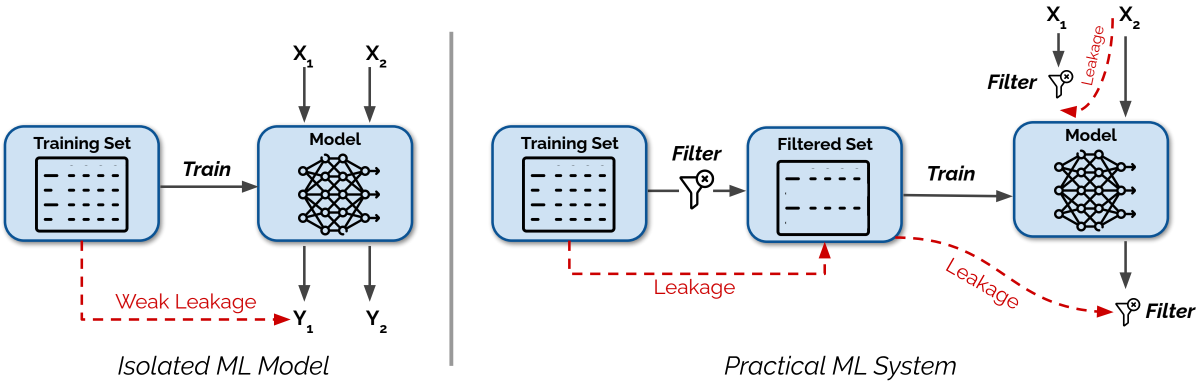

Figure 1: Most past work studies the privacy leakage of ML models in *isolation*. However, in reality most models are part of larger *systems* that contain components for filtering training data, blocking certain test inputs, monitoring model outputs, and more. In this work, we show that adaptive adversaries can create side channels that exploit these system-level components to significantly amplify privacy leakage.

- **Query filtering (Section 6).** Many ML systems use test-time query filters that reject certain inputs, e.g., detectors for adversarial examples [48, 51, 65] and model extraction attacks [41, 49, 57, 85]. We show that since many of these filters aggregate information across different users (to protect against Sybil attacks), adversaries can reveal information about other users' queries by sending targeted inputs. Unlike the previous side channels that reveal private *training* examples, this approach reveals other users' private *test* queries, which is otherwise impossible when considering isolated ML models.

Taken together, our results highlight the need to shift to a system-level view when analyzing ML privacy. Unfortunately, we show that guaranteeing system-level privacy is challenging, both empirically and theoretically, due to highly non-trivial interactions among different components. For example, we show that combining DP and deduplication (as suggested by past work [60]) leads to worse privacy than DP on its own. Furthermore, system-level analyses require tackling the trade-offs that arise when we consider a model's trustworthiness across *multiple* axes, e.g., query filters that improve security can simultaneously hurt privacy. Overall, we hope that our work can contribute to future improvements on the frontiers of ML privacy.

2 Preliminaries

2.1 Background and Related Work

Our paper studies a wide range of attacks that exploit numerous ML models and system components. In this section, we discuss general preliminaries for ML systems and privacy attacks; we otherwise defer the relevant background to each self-contained attack section (Sections 3–6).

2.1.1 Standalone ML Models

An ML model is a function $y \leftarrow f(x)$ that is trained on a dataset D_{train} and evaluated at inference time on a sequence of queries $D_{\text{test}} = \{x_1, x_2, \dots\}$. Most past work on ML privacy considers a "standalone" or "isolated" ML model where f is trained directly on D_{train} (e.g., using gradient descent) and then independently queried on each data point $x_i \in D_{\text{test}}$ to get outputs $y_i \leftarrow f(x_i)$.

Privacy attacks and defenses. When considered as standalone functions, ML models can memorize and leak considerable information about their training data. Numerous attacks exploit this leakage, including *membership inference attacks* [16, 22, 71, 84] that infer whether a specific example was in the training set, and more powerful *data extraction attacks* [5, 14, 15, 17, 54, 72] that can reveal entire training examples. To defend against these attacks, models can be trained with *differential privacy* [27]—typically by clipping and adding noise to updates in stochastic gradient descent [1, 23]—which makes them provably secure against privacy leakage. Other more heuristic approaches such as *data deduplication* [42, 47] or *output filtering* [31] can also mitigate some (but not all) memorization [18, 40, 42].

Membership and non-membership inference. Many of our side-channels exploit membership inference attacks, where the goal is to infer whether a specific data sample x was used in the training of a model f. In real-world scenarios, it is important for attackers to have low false-positive rates as most arbitrary samples are non-members [16]. In addition, inferring *non*-membership is significantly easier than inferring membership, since ML models typically make no mistakes on their training data (i.e., if a model gets an example wrong then it is probably a non-member). For many of our attacks, we show that side channels can be used to convert a membership inference problem into an equivalent (and easier) non-membership inference problem.

2.1.2 ML Systems

We consider part of an ML *system* everything that composes the data collection, training and deployment pipeline. These components augment the standalone model in Section 2.1.1 with additional components that act upon model inputs or outputs. In particular, we consider the following generic system components (see Figure 1 for an illustration):

- **A training data filter** (Section 3) is a function that modifies or removes inputs from the data before training. Popular examples include outlier removal or deduplication.
- **An input pre-processor** (Section 4) is a function that modifies inputs to prepare them for running through the model. Popular examples include image cropping, or tokenization and truncation for language models.
- **An output post-processor** (Section 4) is a function that acts on the output $y \leftarrow f(x)$ produced by the trained model and returns a modified output y' (possibly an empty output $y' = \perp$). Most relevant to our work are memorization filters that reject outputs y that leak information about some training example.
- **A query filter** (Section 6) is a function that acts on the sequence of input queries D_{test} sent to a trained model and returns modified inputs $D'_{\text{test}} = \{x'_1, x'_2, \dots\}$. The function may reject some input queries (i.e., $x'_i = \perp$) in which case the system returns no output for that query. Common examples include methods for rejecting inputs that are perceived as being part of some attack attempt.

Privacy side channels in ML systems. We broadly define a privacy *side channel* in an ML system as any instance where an attacker can probe the system-level components to leak significantly more information about the training set D_{train} or query set D_{test} than what a similarly capable attacker could leak by interacting with a standalone model. This additional leakage is made possible when instantiations of the above filters introduce *dependencies* between different model inputs—either at training or inference time.

We do not consider modifications to the training algorithm [73] or to the training data [77] to be "side channels" as they also apply to the isolated model case. Similarly, other types of attacks such as exploiting ML models used for computer systems [68] (e.g., caches, databases, etc.) are out-of-scope for our study.

We note that prior work has considered the impact of system components on the *integrity* of ML model deployments, e.g., by abusing image preprocessing algorithms to create adversarial examples [61, 82].

2.1.3 A Comparison to Physical Side Channels

In cryptography and computer security, side-channel attacks are typically defined as attacks that exploit leakage from the *implementation* of a protocol or algorithm, rather than from the design itself. Side-channel attacks often rely on leakage through physical phenomena (e.g., power consumption or timing [44, 45]) or auxiliary functionalities such as error messages [11, 78]. In the context of machine learning, physical side-channel attacks can recover the weights or architecture of standalone ML models, e.g., via electromagnetic signals [6], timing [26], power consumption [81], GPU context-switches [79], memory access patterns [37], fault injections [12], or cache timing [35, 83]. In addition, one can reconstruct model queries at inference time using power analysis [80].

Our work considers a different form of side channel that does not rely on physical signals, but instead relies on leakage introduced by auxiliary functionalities present in real-world deployments of machine learning. This is analogous to how deploying secure encryption alongside functionalities like error reporting enables new side channel attacks [11, 78]), or how deduplication can harm cloud storage systems [33]. In contrast to physical side-channel attacks, our attacks do not require any physical access or fine-grained measurement of the ML model. Instead, we only require black-box access to the system's prediction function.

2.2 Threat Model

Adversary's goal. The adversary's goal is to leak information about the training data D_{train} or test queries D_{test} of an ML system by leveraging system-level side channels. For membership inference attacks, we consider the standard adversarial game where we pick a data point x, include this sample in D_{train} with 50% probability, and then have the attacker predict whether x was in D_{train}. We also consider test-time membership inference attacks where the attacker has to guess whether some query x has previously been made to the system or not, i.e., if $x \in D_{\text{test}}$.

Adversary's system access. We assume that the attacker has only black-box access to the ML system, i.e., they can send arbitrary queries and observe outputs, but they do not know the exact weights of the ML system. On the other hand, we assume the attacker knows which system components are used (e.g., output filters) and how they are implemented (for some of our attacks, this knowledge is not strictly necessary). We believe this assumption is reasonable for three reasons: (1) we want to distinguish privacy that may be inherent in a system, from "privacy-through-obscurity" that arises solely due to (likely short-lived) secrecy of some system configurations; (2) many system components we consider have a small number of standardized implementations that practitioners are likely to use. For example, imagededup[1] and DataTrove[2] are the most widely used implementations for data deduplication of images and text, respectively, and offer only one or two canonical

[1] https://github.com/idealo/imagededup

[2] https://github.com/huggingface/datatrove

methods; (3) some of our attacks are reasonably robust to uncertainty and work even without precise knowledge of the system: for instance, our attack against GitHub Copilot works even though the system is a black-box.

In some of our attacks (Section 3, 5), we assume that the attacker can poison the training set, roughly mirroring the setting of Tramèr et al. [77]. In our data extraction attack against language models (Section 4.1), the attacker is assumed to know the prefix p of a string $p||s$ from the training set. For all other attacks, we make no additional assumptions on the adversary.

Since we are introducing a new class of attacks, we generally assume that there are no explicit defenses in place against privacy side channels. However, many of our attacks operate in settings where a model developer has deployed provable or heuristic defenses against privacy leakage (e.g., differential privacy, data deduplication, or memorization filters).

2.3 Ethics and Broader Impact

The attacks that we present could present a threat to user privacy in deployed ML systems. To mitigate these harms, we either study systems that we create ourselves using public data (e.g., CIFAR-10 classifiers) or ones such as GitHub Copilot that are trained on publicly-available data and thus do not pose any real-world privacy risks. The goal of our work is to bring to light these privacy vulnerabilities in order to spur future work on developing private end-to-end systems.

3 Attacking Training Data Filters

We first study side-channel attacks on training data filters. We show that common data filters introduce strong *co-dependencies* between data samples: a data point might be filtered out if and only if some other data points are present in the training set. This introduces a side channel where an adversary can determine, with high confidence, whether a target data sample was present in the training set.

To isolate the privacy leakage that stems from the addition of a specific filter, we assume throughout this section that the system applies a single data filtering scheme before training.

3.1 Attacking Training Data Deduplication

Most large-scale ML models are trained on data scraped from the Internet [30, 67]. Due to the nature of Internet content, it is common for some training data instances to be repeated multiple times; this is problematic for several reasons, including that repeated training data is much more likely to be memorized [18]. The process of *deduplication* addresses this challenge by removing exact or near-duplicated samples [47]. Most modern large language models and image generation models use training data deduplication, e.g., Gopher [63], DALLE-2 [64], LLaMA 2 [75], PaLM 2 [2], and others.

Types of data deduplication. We primarily focus on deduplication applied to images, although the principles underlying our attacks are directly applicable to other domains such as text, as we show in Section 3.1.5. As in Lee et al. [47], we consider both *exact* and *approximate* deduplication. Exact duplicates are samples x, x' that are equivalent, i.e., $x = x'$. Approximate duplicates are samples x, x' that are "close", i.e., $\mathsf{sim}(x, x') \geq \alpha$ for some similarity measure sim and threshold α. Once duplicates are identified, there are two approaches for deduplication: *delete-all* and *delete-all-but-one*. If $x \in D_{\text{train}}$ has as least one duplicate $x' \in D_{\text{train}}$, then *delete-all* removes all duplicates x, x'. *Delete-all-but-one* keeps exactly one copy, i.e., only one of x' and x is retained. While the latter may seem more natural, the former is simpler to implement; both approaches have been proposed in prior work and used in practice [47].

The side channel intuition. Data deduplication introduces co-dependencies between data points: a sample x is deleted if and only if some similar sample x' is present in the training set. An attacker can exploit this side channel to perform a strong targeted membership inference attack. Concretely, suppose the attacker wants to infer membership of some sample x and knows that some near-duplicates x' are in D_{train}. Then, the adversary can infer if x is in D_{train} by detecting whether the duplicates x' were deleted or not. Compared to a standard membership inference attack on the targeted point x, this side channel amplifies the privacy leakage if the duplicates x' known to the adversary are more likely to be memorized than the original data point.

3.1.1 Our Attack

We operate in the same threat model as Tramèr et al. [77]: the attacker can *poison* a small fraction of the dataset to leak information about other data points. In our case, the adversary poisons the training set by inserting *duplicates* x' of a targeted data point x. We assume the attacker has black-box query access to the trained model f and they know the data deduplication procedure that is used.

We focus on classification tasks with labeled data (x, y) and extend our attack to language modeling in Section 3.1.5. We assume that deduplication is independent of a sample's label, i.e., two points (x, y) and (x', y') are duplicates if x is close to x', even if $y \neq y'$. As in Tramèr et al. [77], we assume the attacker can introduce mislabeled data points into the training set. This assumption could be relaxed in some settings by considering clean-label poisoning attacks [70]. We further assume that the model is trained on all (deduplicated) data (i.e., there is no data subsampling step).

We propose attacks for the four forms of deduplication that we introduced above:

- *Exact deduplication, delete all*: Given a target (x, y), we add a mislabeled duplicate (x, y') into the dataset. We then run *non-membership* inference on the mislabeled point: if

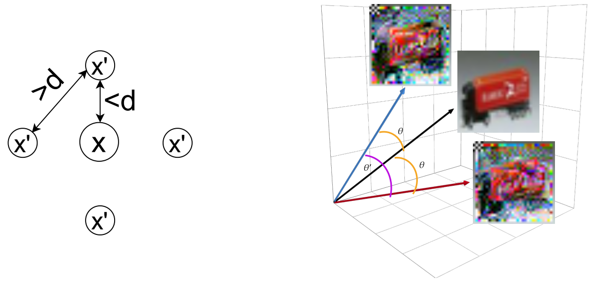

Figure 2: A depiction of our "hub-and-spokes" attack on data deduplication. **Left:** we insert poisoned examples that are each close to the "hub" (x) but are far from each other. **Right:** actual images from our attack. We also include a checkerboard backdoor in the top-left corner of each near-duplicate image to enhance memorization.

this point is absent, we know that the target was present before deduplication. This attack only requires inserting a single poison example.

- *Exact deduplication, delete all-but-one*: In this setting we still use the same attack as above, but the attack is less powerful: when the target is a member, the poisoned duplicate only gets removed with 50% chance. This attack also only requires inserting a single poison example.
- *Approximate deduplication, delete all-but-one*: This is the most complex and interesting attack setting. We create N approximate duplicates $(x'_1, y'), \dots, (x'_N, y')$ in a "hub-and-spoke" pattern (described further below and illustrated in Figure 2). This causes the attacker's points to be near-duplicates of the target, i.e., $\mathsf{sim}(x, x'_i) \geq \alpha$ for all i, but not of each other, i.e., $\mathsf{sim}(x'_i, x'_j) < \alpha$ for all $i \neq j$. The attacker then runs a non-membership inference attack across *all* N poisoned samples x'_i. In this way, the attack becomes stronger as the attacker introduces more poison examples.
- *Approximate deduplication, delete all*: We repeat the same approximate deduplication attack, but the attack is stronger as all examples are removed.

We observe that the side-channel cannot be trivially removed via dataset pre-processing. Manual filtering is too impractical for modern, large datasets, and automated filtering would not be able to remove the side-channel: a filter removing equal samples with differing labels would lead to a scenario equivalent to the *delete all-but-one* deduplication described above. Also randomly subsampling the training data would not be enough to defend from our attack against approximate deduplication with multiple poisons (even though it might be slightly weaker). However, the attack with a single duplicate may fail.

Generating approximate duplicates. In our experiments, we use a standard approximate deduplication procedure based on the cosine similarity between the embeddings $h(x), h(x')$ of two samples, where h is a neural network. Assuming that h outputs embeddings of unit norm in $\mathbb{R}^d$, we have $\mathsf{sim}(x, x') := h(x) \cdot h(x')$. We thus want to find duplicates $x'_1, \dots, x'_N$ of a target x such that $h(x) \cdot h(x'_i) \geq \alpha$ for all i, and $h(x'_i) \cdot h(x'_j) < \alpha$ for all $i \neq j$. To find approximate duplicates, we first compute embeddings that satisfy the above condition, and then "invert" the embedding function h to find input images.

Assume without loss of generality that the target embedding $h(x)$ is the first basis vector in $\mathbb{R}^d$, i.e., $h(x) := (1, 0, \dots, 0)$ (for a general unit-norm embedding $h(x)$, we simply have to apply an appropriate rotation to all vectors). We now build $d-1$ unit vectors $e_1, \dots, e_{d-1}$ of the form

$$e_i := (\alpha, \underbrace{0, \dots, 0}_{i-1}, \sqrt{1-\alpha^2}, 0, \dots, 0) \; .$$

This ensures that: (1) all the embeddings e_i are of unit norm; (2) the cosine similarity between each embedding e_i and the target $h(x)$ is α; (3) the cosine similarity between each pair of near-duplicate embeddings is $\alpha^2 < \alpha$. Given these embeddings $e_1, \dots, e_N$, we invert the embedding function h by running 1000 steps of projected gradient descent. That is, we optimize the input image x'_i so as to maximize the cosine similarity between $h(x'_i)$ and e_i.

To further maximize the membership inference signal, we make the mislabeled near-duplicates (x'_i, y') to be as easy to memorize as possible. We achieve this by adding a common backdoor feature to all near-duplicates (a checkerboard pattern as in Gu et al. [32]). In the extended version of this paper [25, Appendix A] we show that including this backdoor pattern significantly strengthens our attack. In Fig. 2, we present an example depiction of our approximate duplicate attack.

Non-membership inference attack. After poisoning, we adapt the LiRA method [16] to run membership inference. In particular, we train "shadow models" that either contain, or do not contain, the attacker's poisoned duplicates. We query all the shadow models to obtain the models' confidences on the duplicates' poisoned label, and then fit two multivariate Gaussians (one when the duplicates are members and one when they are not) on the model's confidence on each duplicate.

For the final attack, we query the target model f on each duplicate to obtain the confidence in the poisoned label, and then perform a Gaussian likelihood ratio test to determine whether the duplicates are likely to be members of the deduplicated data. If the duplicates are predicted as non-members of the deduplicated data, then we predict that the target example x was in the original training set.

3.1.2 Evaluation

We evaluate our attacks on standard models trained on CIFAR-10 and deduplicate the images using the popular open-source imagededup Python library. We use the library's default settings, which compares images by the cosine similarity of their embeddings computed from a pretrained MobileNetV3

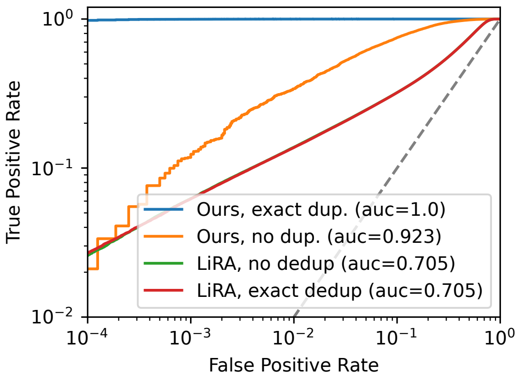

Figure 3: **Deduplication can significantly worsen privacy.** We show membership inference effectiveness under both exact deduplication (*delete all*) and no deduplication. With deduplication, the side-channel leads to near-perfect membership inference; without, it is similar to the baseline poisoning-aware Truth Serum attack [77]. The LiRA baseline [16] performs similarly in both cases.

model [36]. Two images are considered duplicates if the similarity is at least $\alpha = 0.9$.

We choose 250 targets at random from the CIFAR-10 training set and create between 1 and 8 exact- or near-duplicates per target. We build the training set D_{train} by randomly sampling 50% of the CIFAR-10 training set and adding the duplicates. We train the models for 100 epochs.

Following Carlini et al. [16], we evaluate the success rate of the membership inference attack by its true-positive rate (TPR) at a low false-positive rate (FPR). As baselines, we consider their original LiRA attack as well as the attack of Tramèr et al. [77] that combines LiRA with data poisoning (similar to our attack, but not leveraging the side-channel).

3.1.3 Results

Deduplication creates a strong side channel that enables near-perfect membership inference. In Fig. 3, we compare membership inference attacks against models trained with and without delete-all, exact data deduplication. On deduplicated data, our attack achieves essentially perfect targeted membership inference, with a TPR of 98% at a FPR of 0.01%. When data deduplication is not applied, our attack still outperforms LiRA because of the amplification that poisoning has on membership inference, as shown by Tramèr et al. [77].

In Table 1, we provide a more comprehensive comparison between our attack and the prior state-of-the-art attack (Truth Serum [77]) under different deduplication settings. Although both attacks leverage poisoning by introducing mislabeled samples, only ours explicitly exploits the deduplication side-channel. For the single poison case, both our attack and Truth Serum add a mislabeled copy of the target to the training set. However, our attack differs in how we perform inference: we perform *non-membership inference* on the *mislabeled* duplicate, while Truth Serum performs membership inference on the correctly labeled target sample. The latter is sub-optimal when deduplication is applied, since the correctly labeled target is *never* in the training set. In the *delete all* deduplication setting, this change boosts the attack's TPR to near 100%.

Table 1: **Our attacks improve prior privacy-poisoning attacks.** We report the TPR at a FPR of 0.1% and 0.01%, and we compare our attacks against Truth Serum [77] and LiRA [16] under different deduplication strategies. In parentheses, we report the number of poisons injected.

		TPR	
Deduplication	Attack	@ 0.1% FPR	@ 0.01% FPR
None	LiRA Truth Serum (1) Truth Serum (8)	6.2% 18.0% **30.5%**	2.6% 4.0% **6.3%**
Exact or Approx *delete all*	LiRA Truth Serum (1) Truth Serum (>1) Ours (1)	6.2% 89.8% 0.1% **99.6%**	2.7% 79.3% 0.0% **98.0%**
Exact *delete all-but-one*	LiRA Truth Serum (1) Truth Serum (>1) Ours (1)	6.2% 41.6% 3.1% **45.6%**	2.7% 31.8% 1.1% **40.9%**
Approximate *delete all-but-one*	LiRA Truth Serum (>1) Ours (8)	6.5% 0.1% **96.0%**	2.9% 0.0% **90.4%**

In the *delete all-but-one* setting, our attack has lower overall TPR (but still outperforms Truth Serum) due to the possibility that one of the poisons survives deduplication even when the target is a member. With approximate deduplication, our hub-and-spoke attack design with 8 duplicates doubles the attack's TPR compared to using a single poison. Note that in contrast to our attack, running Truth Serum with >1 poisons completely breaks the attack when deduplication is applied. This is because Truth Serum's poisons are all exact (mislabeled) copies of the target, and thus, deduplication removes the poisons regardless of whether the target is in the training set or not. Our attack sidesteps this issue by ensuring that the attacker's poisons are near-duplicates of the target, but *not* near-duplicates of each other.

3.1.4 Dealing with Attacker Uncertainty

In practice, an attacker might not know exactly which deduplication method is being used, or with which hyperparameters (for approximate deduplication). Here, we show that an attacker can still implement a successful attack in such a case.

Unknown deduplication method. In this case, the attacker should apply our attack that adds one mislabeled exact duplicate. Note that this attack is assumption-free: it requires no knowledge of the deduplication approach that is used. This strategy is optimal for three-out-of-four deduplication settings: Exact *delete all*, Approx *delete all*, Exact *delete all-but-one*. The one setting where this attack is not optimal is for the Approx *delete all-but-one* setting. However, in this case the attack with one mislabeled exact duplicate still outperforms Truth Serum, the prior state-of-the-art (the results are the same as for the Exact *delete all-but-one* setting, where our attack has a 9% higher TPR @ 0.01% FPR compared to Truth Serum, as shown in Table 1).

The attacker knows Approx *delete all-but-one* is used. If the adversary knows that the system is using Approx *delete all-but-one* deduplication, and they can guess the α reasonably well, then they can improve over the assumption-free attack above. If the attacker's guess for α is close to correct, the attack is near-perfect (96.0% TPR @ 0.1% FPR) by adding eight approximate duplicates. Specifically, the attacker's guess must satisfy $\alpha^2_{guess} < \alpha_{real} \leq \alpha_{guess}$. If the attacker's guess does not satisfy the above, or if the deduplication method is not actually Approx *delete all-but-one*, the attack fails. Thus, if the attacker is not confident they can guess α correctly, or they are uncertain if Approx *delete all-but-one* is used, they should just use the assumption-free attack above, and still outperform Truth Serum.

How can the adversary guess α?. The adversary can estimate α by computing a TPR-FPR curve over a sample of training data. We estimate that α is unlikely to be smaller than 0.8 in practice as this would yield too many false positives, and reduce the size of the training set. Moreover, the attacker does not need to guess α *exactly*. Even if the guess is off by ±2%, most of the poisons will survive the deduplication process, as shown in Figure 4. Crucially, this rules out α randomization as a potential defense against our attack. Finally, if the same deduplication method is reused multiple times, the attacker can recover the real alpha value by adding dummy near-duplicates with different similarities. Thus, if a developer collects multiple training sets over time and keeps the same deduplication process, the attacker can learn the parameters once and then use this to reliably attack future training runs.

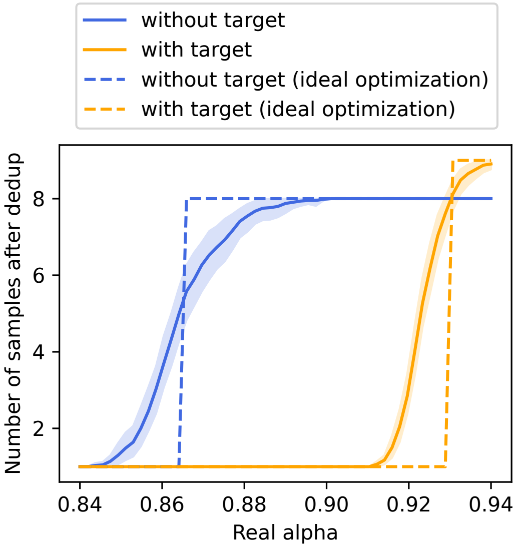

Figure 4: **Our attack is robust to uncertainty in the deduplication threshold α.** We create approximate duplicates by optimizing for the poisons to be less than α^2-similar to each other, and more than α-similar to the target. We show that, within a ±2% range, six duplicates are kept on average when the target sample is not a member–with only one sample being kept when the target sample is a member.

3.1.5 Extension to the Text Domain

While we have focused on classification tasks so far, our attack techniques can easily be extended to other tasks such as language modeling. Text data is commonly deduplicated by finding and removing substrings of k tokens that are exactly repeated across documents [47]. We show this method creates a side channel that enables a powerful *attribute inference attack* (a more powerful attack than membership inference). Specifically, we show that the attacker can infer a low-arity attribute (e.g., a medical condition) contained within a known piece of training data (e.g., a template of a healthcare form). For simplicity of exposition, we assume the unknown attribute is a single token, but our attack could be generalized to longer attributes. We assume the training set contains a sentence of the form

$$\underbrace{S_1,\dots,S_{k-1}}_{\text{pre}} \; \underline{X} \; \underbrace{S_{k+1},\dots,S_{2k-1}}_{\text{post}} ,$$

where pre and post are a prefix and suffix of lengths $k-1$ known to the attacker, and $\underline{X}$ is an unknown token that comes from a set of N possible values $\{\mathsf{X}_1,\dots,\mathsf{X}_N\}$. The attacker's goal is to infer the value of $\underline{X}$.

Our attack. At a high level, our attack creates N families of poison strings (with k poison strings per family), and where each family's strings contains some "canary" substring that is deduplicated if and only if the true value of the unknown attribute is X_i. Then, by running an inference attack to detect the presence of these canary substrings, the attacker can infer which attribute value is correct.

More concretely, we build the i-th family by choosing some unique strings A_i, B_i (each of length less than k) and building k strings of the form:

$$\begin{array}{lll}
A_i + S_1 S_2 S_3 \dots S_{k-1}\, \mathsf{X}_i & & + B_i \\
A_i + \quad S_2 S_3 \dots S_{k-1}\, \mathsf{X}_i\, S_{k+1} & & + B_i \\
A_i + \qquad S_3 \dots S_{k-1}\, \mathsf{X}_i\, S_{k+1} S_{k+2} & & + B_i \\
 & \ddots & \\
A_i + & \mathsf{X}_i\, S_{k+1} \dots S_{2k-1} & + B_i\,.
\end{array}$$

Our construction ensures that: (1) none of the adversary's $N \cdot k$ poison strings contain a duplicated substring of length k; (2) if the i-th family of poison strings has the correct value of the unknown attribute $\underline{X}$, all strings in the family are deduplicated to the string $A_i + B_i$ (we assume $A_i + B_i$ to be shorter than k so that it is not deduplicated recursively).

Once the language model has been trained on deduplicated data, the adversary can compute the model's loss on each of the N strings $A_i + B_i$. The string with the lowest loss likely corresponds to the index of the unknown attribute, as these strings only appear contiguously in the training set if deduplication has occurred.

3.1.6 Deduplication Is Still Worth It in Practice

Deduplication is a cornerstone in mitigating memorization in deployed ML models trained on web-scale datasets [42, 47]. Moreover, differentially private training is only effective on deduplicated data (or else privacy leakage scales exponentially with the number of duplicates). However, if deduplication can induce *new* privacy violations as we show above, should practitioners still apply it? We argue yes, because the empirical cost of *not* doing so is just too high—the privacy risks of duplicated data are far too egregious. This highlights an unfortunate trade-off, where the best practice of using deduplication will be vulnerable to adversarial attacks, and we thus hope that new techniques can be developed to deduplicate datasets without aggravating privacy leakages.

3.2 Attacking Poisoning Defenses

We next look at a broader class of data filtering techniques that aim to protect against poisoning attacks [7]. Poisoning attacks introduce malicious behavior into a model, often by adding many samples that share a common feature (e.g., a backdoor [21]). Defenses against poisoning often work by removing examples based on their relationships to other examples and thus they create "codependencies" that can be targeted to leak information.

3.2.1 Poisoning Defenses

Defenses against poisoning attacks find and remove clusters of points that are part of a presumed attack. Many defenses end up performing some form of data deduplication and are thus vulnerable to a similar side channel as presented in the previous section.

As a straightforward example, Carlini [13] propose a defense against data poisoning that detects poisoned samples and also occasionally flags near-duplicates (as false-positives). Thus, deploying this defense would immediately create the exact same privacy side-channel as described in the previous section.

Alternatively, Sybil filtering [4, 29, 56] is a common countermeasure to prevent colluding adversaries from poisoning a federated learning (FL) model [50]. Most implementations of Sybil filtering downscale or remove a client's model updates if they are too similar to another client's updates. The defense's intuition is that colluding clients performing a poisoning attack are likely to contribute very similar updates, while benign clients' updates are likely to be diverse due to the *non-iid* nature of their datasets. This defense essentially deduplicates at the model-update level rather than at the training-example level.

Finally, in the extended version of this paper [25, Appendix A], we discuss another related poisoning defense, *activation clustering* [19] that is also vulnerable to similar side channels.

The side channel. The attacker introduces malicious samples so that the poisoning defense is triggered if and only if some target is present. This yields a membership inference side channel. Depending on the defense, this side channel can leak information about individual training samples or collections of samples (e.g., in the FL case when models updates represent information about a batch of examples).

3.2.2 Our Attack

For the semi-supervised poisoning defense of Carlini [13], the attack is exactly the same as for general data deduplication, so we do not discuss it further. We thus focus on Sybil filtering in FL, which introduces some notable differences into the attack since the defense operates over model updates instead of individual samples.

We assume a threat model where the attacker controls one FL client and knows the defense mechanism being used. The attacker's goal is to infer if some target client is participating in the protocol (i.e., client membership inference). We assume the attacker knows the target client's data distribution (e.g., if we are training a model on people's photos, then the attacker has some public photos of the target client). We do not assume that the attacker's data and the target's data overlap—only that they are similarly distributed.

We consider a canonical defense called FoolsGold [29], which dynamically modifies the learning rate of individual clients based on the similarity of their updates over time. To compare the updates of two clients, the defense aggregates each client's history of updates and then computes the cosine similarity between these aggregates. A "Sybil" score between 0 and 1 is then computed for each client—based on the maximum similarity with another client—and this score is used to adjust the contribution of that client's model updates.

Our attack adds a malicious client into the training pool, with the aim of producing model updates that are similar to those produced by the target (if present). The attacker then performs membership inference based on the model's loss on the *attacker's* (not victim's) data: if the target client is not participating in the protocol, FoolsGold will not downscale the attacker's updates and so the loss will decrease across epochs; conversely, if the target is present, the attacker's and target's updates are both downscaled and so the loss *will not decrease*. The attacker thus measures the model's loss on the attacker's dataset before and after participating in he FL protocol to determine if the target is present.

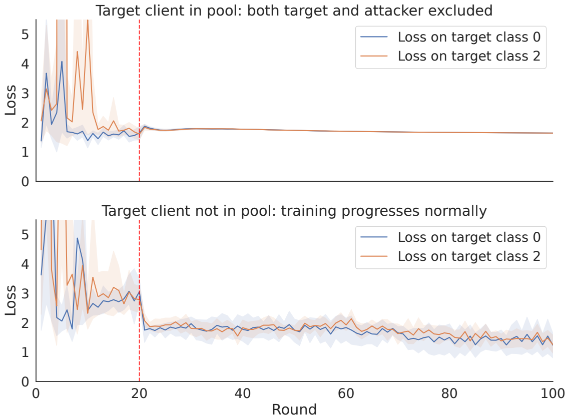

Figure 5: **When federated learning is combined with defenses against data poisoning, a side-channel is opened that worsens privacy.** We run the FoolsGold defense and insert a poisoning client into the learning protocol (denoted by the red vertical line). When a target client of interest is present in the data, both that client and the poisoning client will effectively not contribute to the learning (top of Figure). When the target is not present (bottom of Figure), the learning on the poisoning client continues as normal. This enables a strong membership inference attack.

3.2.3 Evaluation

We experiment with a non-iid setup that simulates a natural FL deployment to train an image classifier on pictures from each user's local device. We use the Labeled Faces in the Wild [38] (LFW) dataset, restricting the classification task to the 5 identities with over 100 pictures, and generate 5 clients each with 90 training images. To enforce the non-iid setup, we sample a majority (80%) of images from a single identity for each client and then fill the remainder with images sampled from other classes according to a Dirichlet distribution ($\alpha = 1$). We train a ResNet-18 [34] using the official FoolsGold implementation.[3] The malicious client joins the protocol at round 20; their dataset is composed of 50 non-overlapping images from the target client's identity. This ensures that its updates will be similar to the targeted client and different from others. We run experiments with two target classes (0 and 2) and repeat each experiment 10 times with different random seeds.

3.2.4 Results

Figure 5 validates the susceptibility of FoolsGold to our side-channel attack. As predicted, the model's loss on the attacker's dataset only decreases when the target is not present. By using the difference in average loss for 15 rounds before and after the attacker participated, we achieve 85% accuracy (averaged over 10 experiments) in identifying whether the target client was in the training pool.

Without the side channel, the adversary would need to make a membership guess based solely on the model's loss on the attacker's data. A common approach from the literature would be to train shadow models. However, this is difficult in an FL setting as the attacker would need to know the data distribution of *all* clients (not just the target), and perform multiple expensive simulations of the entire FL run. We simulate such a strong adversary by taking the 10 different runs of our experiment, and setting the loss threshold directly as the difference between the model's average loss when the target is participating, or not. We find that even such a strong adversary only achieves 67.5% accuracy in guessing the presence of the target, without relying on our side channel.

4 Attacking Input and Output Filters

Here, we consider how filters applied to a model's inputs or outputs can enable powerful membership inference attacks. In fact, these membership inference attacks will be so strong (essentially perfect) that we can convert them into *data extraction attacks* that even allow extraction of OpenSSH private keys contained in a language model's training set.

A common theme of this section is that ML system filters make it *impossible* for the system to produce certain outputs, depending on the training data. The attacker can then make some guess about an input in the training set, and craft a query that would trigger an impossible output if the guess were correct. This yields a perfect *non-membership* inference attack: if the model produces the impossible output, the attacker knows their guess is incorrect. We then introduce various approaches to lift the attack into a strong (or even perfect) membership inference attack. This attack strategy is reminiscent of impossible differential cryptanalysis [9, 43], where an attacker's guess about the internal state of a cryptosystem can be ruled out if certain statistical output properties are observed.

Background: language models. Large language models (LLMs) are neural networks that take as input a sequence of text of variable length and predict a probability distribution over the next word (or token, see below) in the sequence. While early language models represented text as a sequence of characters or words [52], all recent language models use a more compact representation that splits arbitrary text into *tokens* that represent entire words, sub-words, or characters. The set of all tokens is called the *vocabulary*.

When applied to an input sequence of tokens $t_1, t_2, \ldots, t_n$, a LLM predicts a probability distribution $\Pr[t | t_1, t_2, \ldots, t_n]$ for the value of the next token in the sequence. To generate text, the model is applied repeatedly to its own output, using a particular *decoding* strategy. In the simplest case of greedy decoding, we repeatedly sample the most likely token given the current input sequence and append this token to the input.

[3] https://github.com/DistributedML/FoolsGold

4.1 Extracting Vocabularies of Language Models

We first show how to use the input filtering stage of language models as a side channel.

Background: Byte-pair encoding. The inputs to modern language models are preprocessed into a series of sub-words (i.e., tokens) and truncated to a maximum context window size. The set of tokens (i.e., the model's vocabulary) is determined by running an algorithm such as byte pair encoding over the training data [69]. At the time of submission, Byte-pair encoding is the standard tokenization algorithm used in all modern large language models. A model's vocabulary can reveal sensitive details of its training data, e.g., the GPT-2 tokenizer [62] contains tokens that represent individuals' Reddit usernames [66]. While this is not a privacy issue in the case of tokenizers trained on public data (like GPT-2's), this may become a concern if a tokenizer is trained or adapted on private data with an unusual vocabulary, e.g., medical records. Finally, many systems such as the Claude chatbot [3] have their tokenizer kept secret.

The side channel. Here, we show that truncating inputs to a fixed size window allows adversaries to extract the entire vocabulary of any model. In particular, consider a sequence such as "My favorite color is red. My favorite color is". Here, most language models will predict the word "red" as the continuation. However, if some padding text of $\geq N$ tokens is added, e.g., "My favorite color is red. PADDING My favorite color is", the model will not be able to see the word "red" for long padding sequences. This allows one to determine how many tokens the padding sequence occupies in the model's input.

4.1.1 Extracting the Complete Vocabulary

In particular, under the byte-pair encoding algorithm, it is guaranteed that any token can be recursively split into two sub-tokens until the final sub-tokens are only one byte. Our attack will leverage this in reverse, where we initialize our extracted vocabulary V to all of the single bytes and recursively expand it. Concretely, for all pairs of tokens $(u,v) \in V \times V$, we query the model on the sentence:

$$\text{“My favorite color is red. } \underbrace{u||v\ u||v\ \ldots\ u||v}_{3N/4 \text{ times}} \text{ My favorite color is”}$$

where $||$ denotes concatenation. Suppose that $u||v$ actually *is* a single token in the vocabulary. Then the number of padding tokens inserted between the question and answer would be just $3N/4$ and the model would respond with "red". In this case we insert the new token $V \leftarrow V \cup \{u||v\}$. On the other hand, if $u||v$ was represented instead by two tokens, then the number of padding tokens would be $3N/2 > N$ and the model will not answer "red".

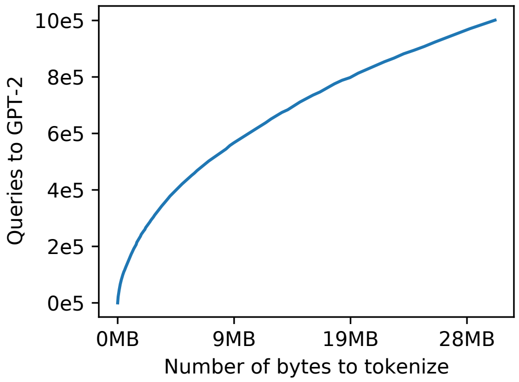

Figure 6: **We extract the tokenizer for GPT-2 on specific byte strings from Wikipedia.** Our attack leverages a side channel based on the fact that language models use of a fixed context window.

Attack complexity. After T iterations of this attack, we will recover all tokens of length less than T. Extracting the entire vocabulary will require $O(|V|^T)$ total queries.

Results. Our attack is empirically effective at recovering the entire GPT-2 vocabulary. In the extended version of this paper [25, Appendix A], Table 3 lists example of tokens that we extract, including rare substrings such as "RandomRedditorWithNo", "TheNitromeFan", and "SolidGoldMagikarp". Thes strings are usernames of Reddit users that were likely repeated frequently in the GPT-2 training dataset.The attack requires 819,869,857 queries.

4.1.2 Targeted Vocabulary Extraction

We also can use this attack to efficiently extract the tokenization for a specific word or phrase. Such an attack may be useful for (1) generating adversarial examples, where it may be useful to know the tokenization of a particular string, or (2) determining if a model uses another model's tokenizer, e.g., for licensing or reverse engineering purposes.

To use the above algorithm for a particular string such as "hello" and "world", we can simply check if "he", "el", "ll", etc. are tokens and would not need to check if "ho", "wl", "ol", etc. are tokens.

Results. In Figure 6 we measure the efficacy of this restricted attack by reporting the number of queries necessary to determine how GPT-2 would tokenize the first N bytes of the enwiki8 dataset (a subset of Wikipedia). On average we find it requires roughly 30,000 queries per megabyte, diminishing as we increase the number of bytes to tokenize.

4.2 Attacking Memorization Filters

We next show how filters applied to LM outputs can also enable aggravated privacy violations.

Background: memorization-free decoding. LLMs are known to memorize—and output—sequences from their training dataset [15]. Thus, popular production LLM systems such as GitHub's Copilot use filters that block outputs that match their training data. In this way, models are guaranteed to *never* emit a verbatim sequence from their training dataset.

Ippolito *et al.* [40] formalize this as *memorization-free decoding*. Memorization-free decoding adds a filter that runs on-line with the language model and, before emitting the next token, checks if this causes a k-gram match to a sequence in the training dataset (using an efficient Bloom filter lookup). If so, this token is replaced with the most-likely non-memorized token instead. While such filters are imperfect (for example they do not prevent the model from outputting similar yet not verbatim copies of training data), they are an efficient and practical defense that reduces the likelihood of data copying.

The side channel. We show that applying any form of memorization-free decoding introduces a significant privacy vulnerability: if a language model *ever* generates any particular k-gram, we are *guaranteed* it was not part of the training data. This gives us a perfect *non*-membership inference attack (i.e., a 100% true negative rate at a 0% false negative rate). Next, we will show how to convert this into a strong *membership inference attack*.

4.2.1 Our Attacks

We propose two different ways of extending the above non-membership inference attack to a near-perfect *true positive rate*, thereby obtaining a near-perfect *membership inference attack*.

Perfect membership inference for toggleable filters. To begin, we first develop a simple counterfactual-based approach for the special case where the memorization filter can be *disabled*. Indeed, some models, such as GitHub's Copilot model, allow the user to **choose** to block the recitation of training data. This gives the adversary the ability to perform a perfect counterfactual analysis.

Concretely, assume the adversary wants to perform (positive) membership inference on the sequence t. We begin by *disabling* the memorization filter and, for each prefix p_i of t, query the model on p_i yielding a predicted suffix s_i. For each of these, we check if $p_i||s_i$ is a prefix of t.

If none of the generations have this property then the attack is inconclusive. However, if we identify *any* index i for which $p_i||s_i$ is a prefix of t, we can *enable* the memorization filter, and again prompt the model with p_i. If the suffix with the filter turned on is equal to the original suffix s_i, then the sequence *cannot* have been in the training dataset—if it was, the filter would have blocked it. On the other hand, if the suffixes are not equal then we know with 100% certainty that the sequence *was* in the training dataset.

Membership inference with a permanent filter. Some models contain memorization filters that are permanently applied, which prevents our straightforward counterfactual approach. The challenge with such a filter is that if a model fails to output some suffix, this could be due to one of two reasons: (1) either the filter was triggered (i.e., the sequence is in the training set); or the model simply assigns a low likelihood to this suffix.

To reduce the likelihood of the latter event, we propose to *encourage* the model to emit the desired string, by exploiting the in-context learning abilities of language models. For example, lets say we want to detect if the sequence "ABCD" is in the training set, and when we prompt the model with "ABC" it fails to output the letter "D". This is either because the memorization filter was triggered, or because the model simply assigns low probability to the completion "D" in this context. To disambiguate these two cases, we prompt the model with the sequence "ABCD ABCD ABCD ABC", where we repeat the targeted completion many times. Empirically, this guarantees that any powerful language model will output the completion "D" (as it is the most likely completion in the given context). Now, if the model *still* fails to output "D", we can assume with high confidence that the memorization filter has been triggered. More formally, we split the target string t into a prefix p_i and suffix s_i, prompt the model on the string $p_i||s_i||p_i||s_i||\ldots||p_i$, and then check if the output is s_i.

4.2.2 Estimating Copilot's Training Date Cutoff

We now describe an application of the toggleable filter side-channel attack to perform membership inference on GitHub Copilot, a popular public coding assistant. Concretely, Copilot uses a toggleable memorization filter that we exploit to determine the date on which GitHub scraped the model's training data. The details of Copilot's memorization filter are not public, and in practice we found that it likely performs approximate filtering based on fuzzy matching between outputs and training data. Nevertheless, we can run our attack as if it was an ideal filter and accept that this could incur false positives (approximate filtering does not cause false negatives).

To evaluate our attack, we choose a popular GitHub repository, `tqdm`, and randomly sample some of the project's commits in which code was introduced that is still present in the most recent revision of the project. We then run our membership inference attack on each of these commits and in Figure 7 we plot the fraction of commits our attack predicted were *not* in the training set.

There are several false positives from before October 2021, but for all commits made after that date, Copilot can generate the code with the filter on. This indicates that they are not members in the training data, and we can infer that October 2021 is likely the date at Copilot's data was last collected. This date is exactly the same date at which other OpenAI models' training data was collected (e.g., `text-davinci-003`),

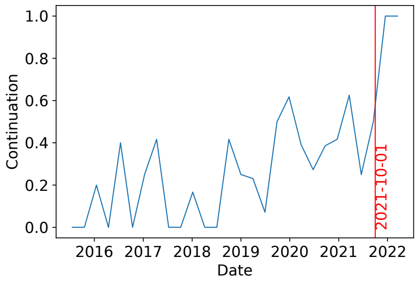

Figure 7: **GitHub Copilot's memorization filter leaks membership information.** The system blocks outputs that are similar to the training data and thus reveals which code was used for training. In particular, we examine code commits that are added to a popular GitHub repository over time and plot the fraction of commits that our attack flags as non-members. There is a noticeable increase in October 2021, which reveals when the training data was last collected.

further confirming this result.

Note that our attack shows that Copilot can generate code for all commits after October 2021 even while the filter is on. The attack does therefore not depend on the filter being toggleable. While in Section 4.2.1 we give a baseline example of how an attack would work if the filter can be toggled, our full attack does not need this assumption. We choose tqdm as a target because it is a popular library with frequent commits, hence providing high resolution for our experiment. However, our attack still works with less popular repositories with less frequent commits. Finally, we attack Copilot's memorization filter as it is the only widely deployed filter we are aware of that matches the whole training corpus. Moreover, since the training data is already public, this experiment poses minimal risk. However, since we have shown a successful attack against a filter with unknown implementation details, we believe our attack should be generally applicable to other systems with a memorization filter.

4.2.3 From Output Filtering to Data Extraction

Finally, we extend our attack to perform a data *extraction* attack that can recover complete documents from the training set token-by-token. We focus on the permanent memorization filter case, and we assume the attacker knows a substring of the text that they wish to extract from the training set.[4] For example, to extract an RSA key the prefix could be "BEGIN OPENSSH PRIVATE KEY".

Given this prefix p of a training document, a perfect membership inference attack makes it trivial to extend the sequence by one token: enumerate all possible tokens s that may occur next and for each token run the attack on $p||s$. In practice, the attack may have false positives and thus identify several next-token candidates s_j^*. These false positives occur when the language model fails to emit a token that was not in the training set even when it is coerced to do so. In these cases, we explore each next-token candidate in a depth-first manner. If a candidate was incorrect, it will simply reach a state where there is no valid next token and terminate.

Table 2: **We can use an output filter side-channel to extract hundreds of secret OpenSSH private keys.** We use GPT-Neo language models with a permanent memorization filter and apply our membership inference approach to iteratively extract single tokens from the training set.

Model	Success Rate	Mean Queries
GPT-Neo 125M	90.3%	376,000
GPT-Neo 1.3B	90.0%	338,000
GPT-Neo 2.7B	89.8%	344,000

Extracting RSA private keys. As a proof of concept, we consider the hypothetical case that GPT-Neo [10] (a family of language models similar to GPT-3) has a memorization filter and was trained on files containing some unknown RSA secret keys. To instantiate the memorization filter, we build a Bloom filter containing all 20-token sequences in the GPT-Neo training dataset and prevent the model from ever emitting these sequences. To simulate training on private keys, we add 1,000 different OpenSSH private keys to the Bloom filter, each of which are 512 base-64 encoded bytes.

Using this setup, we found that our membership inference attack achieves over 99.9% accuracy at predicting whether a next-token candidate is in the training set. However, this is still not enough for accurate data extraction as there are over 10,000 possible next tokens in Base64 output and we want to extract text that contains many such tokens.

To further enhance the attack, we count the frequency of each Base64-encoded token in a large 10TB dataset of random data. We then bias the search towards more common tokens, e.g., the Base64 token "Q" occurs much more frequently than "omanip" in practice (cfr Figure 9 in the extended version of this paper [25, Appendix A]). Using this approach, we query the model with the complete header ("BEGIN OPENSSH PRIVATE KEY") and run our attack. The attack extracts about 90% of the keys successfully and requires around 340,000 model queries (see Table 2).

5 Breaking Differentially Private Training

We next show how the side channels that we have introduced thus far can violate "provable" privacy guarantees when those guarantees are based on isolated ML models. Specifically, we show that standard differential privacy analysis fails when data filtering components are added.

Background: differentially private training. An ML algorithm is differentially private (DP) if the distribution over

[4] We assume the substring is a prefix, but one can extend the attack to work with a known suffix by running membership inference attack on $s||p$.

possible trained models is close for any two neighboring training datasets that differ in a single example [1, 27]. A common paradigm to design DP algorithms is to estimate the algorithm's *sensitivity*—the maximum change in output from changing one input—and then adding random noise calibrated to that sensitivity. However, estimating the sensitivity of a full ML training pipeline is hard. Instead, the approach of the common algorithm DP-SGD [1] is to estimate the sensitivity of each training step and rely on DP's composition property to compute the model's final privacy budget ε.

The side channel. The theoretical analysis of DP-SGD does not consider additional components of the ML training pipeline such as filters applied before and after training. When these filters depend on the training data, they themselves have a non-zero sensitivity (i.e., changing one training example could change the filter's output). As a result, the total sensitivity of the system may be much larger than the sensitivity of the training algorithm in isolation. This means that an ML system trained with DP-SGD could *violate* the provable privacy guarantees that apply to an isolated model.

Warm-up: memorization filters have unbounded sensitivity. As a simple example, consider the memorization-free decoding filter (Section 4.2) that prevents the model from emitting training data. Its sensitivity is unbounded: the filter activates on a given output if and only if some training example matches it. It is thus obvious that training the model with DP-SGD does not prevent the side-channel attacks that we describe in Section 4.2.

Deduplication has large sensitivity. A less obvious example is the case of data deduplication (Section 3.1). One may expect that since deduplication empirically minimizes memorization, combining it with DP-SGD should lead to an extra layer of privacy protection. Indeed, prior work has even suggested this [60]. However, as our attacks in Section 3.1 show, deduplication can cause a single training example to influence whether many other examples get removed. In turn, the sensitivity of deduplication is quite large in practice.

Evaluation. We apply the side-channel attack from Section 3.1 to an ML system that first deduplicates the training set and then trains a model with DP-SGD. We use the CIFAR-10 dataset and pick one training sample at random as the target. We create 256 approximate duplicates of the target and add these to the training set. We then deduplicate the training set using the default imagededup settings as in Section 3.1. We train 128 models using DP-SGD on either the full dataset as described above, or a neighboring dataset with the target removed. Note that after deduplication, these initially-neighboring datasets now differ in 256 examples. We use the state-of-the-art differentially-private training approach from De et al. [24], with a batch size of 4096, augmentation multiplier of 16, and their settings for the noise multiplier and learning rate.

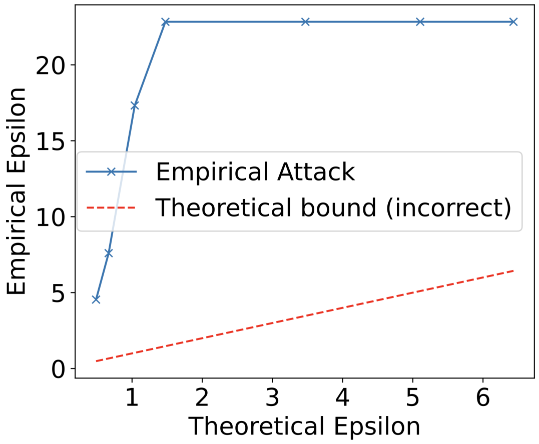

Figure 8: **Deduplication violates naive differential privacy analysis.** If we combine approximate deduplication with DP-SGD (as suggested in prior work), then a single example in the collected training data can affect multiple samples in the filtered training dataset that DP-SGD gets applied to. Deduplication thus effectively amplifies the sensitivity of the differentially private mechanism, leading to a severe underestimate of the true privacy loss of the end-to-end system.

We run DP-SGD multiple times with a target privacy budget $\varepsilon \in (0.5, 6)$ and set $\delta = 10^{-5}$. We then perform a membership inference attack on the adversarial duplicates (as described in in Section 3.1). We use the attack's FPR and TPR to estimate an empirical lower bound on the *true* value of the privacy budget ε for the entire ML system—following prior work on DP auditing [53, 55].

Results. The privacy analysis of DP-SGD severely underestimates the privacy leakage of the entire system (Figure 8). For example, when DP-SGD's theoretical bound on epsilon is respectively 1, 1.5, and 3, the actual privacy budget consumed by the system is *at least* 7.5, 17, and 22. Note this is not due to a mistake in DP-SGD's analysis or a bug in our implementation. Rather, it is because the privacy analysis is with respect to the dataset that DP-SGD is actually run on and not the system's "true" training set D_{train} that is collected prior to deduplication. Thus, computing a correct privacy guarantee for the entire system would require composing DP guarantees for both the deduplication step and the training process.

6 Leaking Test Queries From Query Filters

Thus far, we proposed side channel attacks that leak *training* examples. Here, we show that side channels can also allow adversaries to identify *test* queries made by arbitrary users. Worryingly, these attacks are otherwise impossible to launch when analyzing ML models in isolation.

Specifically, we focus on ML "query filters" that restrict or flag inputs in order to prevent attacks. These filters often guard against Sybil attacks (i.e., where an adversary uses multiple accounts to conceal their attack) by aggregating queries across *all* users of the system. Here, we show that this introduces a side channel, where attackers can create special queries whose outputs reveal information about other users' queries.

6.1 Attacking Stateful Evasion Detectors

Background. In an evasion attack [8, 74], an attacker perturbs an example to cause a ML model to misclassify it. To generate these adversarial examples in a black-box setting, existing query-based attacks repeatedly query the target model [39]. Given the difficulty in outright defending against evasion attacks, recent work has developed *stateful* detection defenses [20, 48]. These defenses record every query (recently) issued to the system, and then ban a user (or take other defensive actions) if their query history appears malicious. Commonly, a sequence of queries is marked as malicious if the queries are close to one another under some similarity metric. As noted above, certain stateful evasion defenses also aggregate queries across users [48].

The side-channel. We target the method of Li et al. [48], the first defense that proposes to store a global history of fingerprints (a fuzzy hash) of each query. If the current user's query is similar to a previous query, an attack is detected and the filter will reject the query and all subsequent similar queries. In turn, this enables a side channel where one can check if another user has uploaded some query x by re-uploading the same query. If the system rejects the attacker's query, they know that a query very similar to x was likely previously uploaded.

Evaluation and results. We evaluate our attack on CIFAR-10. We create a set D_{test} of 1,000 images to simulate queries made by users and a held-out set $D_{\sim\text{test}}$ of 1,000 other images. We query the model on each image in D_{test} and $D_{\sim\text{test}}$ and predict which images were real queries (i.e., D_{test}). Our attack has perfect accuracy (100% TPR, 0% FPR): the defense rejects the attacker's query if and only if the image was previously queried. The only possible failure case for the attack is a false positive, which the defense is explicitly set up to minimize and is thus rare in practice. Overall, this side channel highlights the strong tension between security and privacy—if one uses a privacy-preserving defense that acts on a per-user basis, they cannot resist Sybil attacks.[5]

6.2 Extension to Stateful Extraction Detectors

Aside from black-box evasion attacks, stateful detectors also exist for model extraction attacks [59, 76]. In extraction attacks, adversaries make many queries to a black-box ML model in order to reverse-engineer a local copy with similar functionality. Existing stateful defenses against model extraction [41, 49, 58, 85] keep track of queries and search for examples that are highly similar to one another or are consecutively out-of-distribution. To date, no extraction methods aggregate across users and are thus all vulnerable to Sybil attacks. However, if they were generalized to use a global query log (as suggested in [48] for the PRADA defense in [41]), then they would become vulnerable to a similar privacy side channel as we described in Section 6.1.

6.3 Extension to Retrieval-based Text Detection

Finally, similar side channel attacks exist for retrieval-based text detectors [46]. Here, the goal is to test whether some generation y was produced by a particular language model. Krishna et al. [46] approach this by recording each output from a language model and then checking whether any text similar to y was previously generated by the model. Unfortunately, such methods allow adversaries to check if some output was produced as a response to another user's query, as acknowledged by [46].

7 Conclusion

In this work, we showed that real-world ML systems can have drastically worse privacy than is otherwise suggested by typical standalone privacy analyses. In particular, we showed how adaptive adversaries can use black-box or white-box system knowledge to exploit side channels in popular ML components. Taken together, these attacks contribute to a rethinking of how practitioners should measure and mitigate privacy risks for state-of-the-art ML systems.

Perhaps most surprisingly, it is actually the introduction of components that are *intended to improve privacy* (i.e., data deduplication and memorization-free decoding) that leads to side-channels that cause privacy violations. Similarly, methods designed to protect against other forms of adversarial attacks (e.g., data poisoning and evasion attacks) can also backfire and induce privacy violations at elevated rates. These findings highlight the inherent tensions between security and privacy, and how improving privacy in the average case can have adverse effects when faced with worst-case adversaries.

Moving forward, there are numerous rich areas for future work in the space of system-level analyses of ML systems. In particular, while we focus specifically on privacy side-channels, our work hints at the possibility that other side-channels may exist for different types of attacks and threat models. Moreover, our work highlights open problems in ensuring privacy-preserving ML, such as how to enforce differential privacy when composing hard-to-analyze ML modules such as filters or anomaly detectors. We hope to tackle these and other challenges in future work.

[5] Note that it may be possible to improve the defense. For example, instead of filtering a query when it is similar to *one* prior query, the defender could pick a random threshold T on the number of similar past queries. We leave an exploration of this to future work.

Author Contributions

- Nicholas and Florian proposed the problem statement of side-channel leakage in ML system components.
- Florian proposed the side channels in data deduplication and stateful detectors.
- Florian and Matthew proposed the side channels in data poisoning defenses.
- Nicholas and Eric proposed the side channel in memorization filters.
- Nicholas proposed the side channel to extract vocabularies of LLMs.
- Edoardo performed experiments on data deduplication and stateful evasion detectors, and drafted the corresponding sections of the paper.
- Giorgio and Matthew performed experiments on data poisoning, and drafted the corresponding sections.
- Nicholas performed experiments on memorization filters and extracting vocabularies, and drafted the corresponding section of the paper.
- Milad performed experiments with GitHub Copilot and on attacking DP-SGD, and drafted the corresponding sections of the paper.
- Edoardo, Giorgio, Nicholas, Milad, and Eric designed the paper's figures.
- Edoardo, Giorgio, Nicholas, Christopher, Matthew, Milad, Eric, and Florian wrote the paper.
- Florian organized the project.

Acknowledgments

We thank Reza Shokri, Jamie Hayes, Andreas Terzis, Vijay Bolina, Helen King, Eve Novakovic, Amanda Carl, Jenna LaPlante, Sanah Choudry, and Daniel Paleka for helpful discussions. We also thank the reviewers and the shepherd for the useful feedback. Eric Wallace is supported by the Apple Scholars in AI/ML Fellowship. Edoardo Debenedetti is supported by armasuisse Science and Technology.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *ACM SIGSAC*, 2016.

[2] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. PaLM 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[3] Anthropic. Model card and evaluations for Claude models. https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf, 2023.

[4] Sana Awan, Bo Luo, and Fengjun Li. Contra: Defending against poisoning attacks in federated learning. In *ESORICS*, 2021.

[5] Borja Balle, Giovanni Cherubin, and Jamie Hayes. Reconstructing training data with informed adversaries. In *IEEE S&P*, 2022.

[6] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSINN: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX*, 2019.

[7] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[8] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *, ECML PKDD 2013*, 2013.

[9] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *EUROCRYPT*, 1999.

[10] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, 2021. URL https://doi.org/10.5281/zenodo.5297715.

[11] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In *CRYPTO*, 1998.

[12] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. Sniff: Reverse engineering of neural networks with fault attacks. *IEEE Transactions on Reliability*, 71(4):1527–1539, 12 2022. ISSN 1558-1721. doi: 10.1109/tr.2021.3105697. URL http://dx.doi.org/10.1109/TR.2021.3105697.

[13] Nicholas Carlini. Poisoning the unlabeled dataset of semi-supervised learning. In *USENIX*, 2021.

[14] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX*, 2019.

[15] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *USENIX*, 2021.

[16] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *IEEE S&P*, 2022.

[17] Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *USENIX*, 2023.

[18] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *ICLR*, 2023.

[19] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[20] Steven Chen, Nicholas Carlini, and David Wagner. Stateful detection of black-box adversarial attacks. In *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*, pages 30–39, 2020.

[21] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[22] Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *ICML*, 2021.

[23] Christopher A Choquette-Choo, H Brendan McMahan, Keith Rush, and Abhradeep Thakurta. Multi-epoch matrix factorization mechanisms for private machine learning. In *ICML*, 2022.

[24] Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

[25] Edoardo Debenedetti, Giorgio Severi, Nicholas Carlini, Christopher A Choquette-Choo, Matthew Jagielski, Milad Nasr, Eric Wallace, and Florian Tramèr. Privacy side channels in machine learning systems. *arXiv preprint arXiv:2309.05610*, 2023.

[26] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*, 2018.

[27] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, 2006.

[28] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM SIGSAC*, 2015.

[29] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. In *RAID*, 2020.

[30] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[31] GitHub. About GitHub Copilot. https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot, 2022.

[32] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[33] Shai Halevi, Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Proofs of ownership in remote storage systems. In *ACM CCS*, 2011.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[35] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. *arXiv preprint arXiv:1810.03487*, 2018.

[36] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *CVPR*, 2019.

[37] Xing Hu, Ling Liang, Lei Deng, Shuangchen Li, Xinfeng Xie, Yu Ji, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Neural network model extraction attacks in edge devices by hearing architectural hints. In *ASPLOS*, 2020.

[38] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, University of Massachusetts, Amherst, 2007.

[39] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *ICML*, 2018.

[40] Daphne Ippolito, Florian Tramèr, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher A Choquette-Choo, and Nicholas Carlini. Preventing verbatim memorization in language models gives a false sense of privacy. In *INLG*, 2023.

[41] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. PRADA: protecting against DNN model stealing attacks. In *IEEE EuroS&P*, 2019.

[42] Nikhil Kandpal, Eric Wallace, and Colin Raffel. Deduplicating training data mitigates privacy risks in language models. In *ICML*, 2022.

[43] Lars Knudsen. DEAL-a 128-bit block cipher. *complexity*, 1998.

[44] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, 1999.

[45] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, 1996.

[46] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*, 2023.

[47] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *ACL*, 2022.

[48] Huiying Li, Shawn Shan, Emily Wenger, Jiayun Zhang, Haitao Zheng, and Ben Y Zhao. Blacklight: Defending black-box adversarial attacks on deep neural networks. In *USENIX*, 2022.

[49] Xinjing Liu, Zhuo Ma, Yang Liu, Zhan Qin, Junwei Zhang, and Zhuzhu Wang. SeInspect: Defending model stealing via heterogeneous semantic inspection. In *ESORICS*, 2022.

[50] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 2017.

[51] Mohammad Maghsoudi Mehrabani, Amin Azmoodeh, Ali Dehghantanha, Behrouz Zolfaghari, and Gautam Srivastava. Proactive detection of query-based adversarial scenarios in NLP systems. In *ACM Workshop on Artificial Intelligence and Security*, 2022.

[52] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

[53] Milad Nasr, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlini. Adversary instantiation: Lower bounds for differentially private machine learning. In *IEEE S&P*, 2021.

[54] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023.

[55] Milad Nasr, Jamie Hayes, Thomas Steinke, Borja Balle, Florian Tramèr, Matthew Jagielski, Nicholas Carlini, and Andreas Terzis. Tight auditing of differentially private machine learning. *arXiv preprint arXiv:2302.07956*, 2023.

[56] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al. FLAME: Taming backdoors in federated learning. In *USENIX*, 2022.

[57] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. ActiveThief: Model extraction using active learning and unannotated public data. In *AAAI*, 2020.

[58] Soham Pal, Yash Gupta, Aditya Kanade, and Shirish Shevade. Stateful detection of model extraction attacks. *arXiv preprint arXiv:2107.05166*, 2021.

[59] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM ASIACCS*, 2017.

[60] Natalia Ponomareva, Jasmijn Bastings, and Sergei Vassilvitskii. Training text-to-text transformers with privacy guarantees. In *Findings of the ACL*, 2022.

[61] Erwin Quiring, David Klein, Daniel Arp, Martin Johns, and Konrad Rieck. Adversarial preprocessing: Understanding and preventing Image-Scaling attacks in machine learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1363–1380. USENIX Association, 8 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/quiring.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[63] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al.

Scaling language models: Methods, analysis & insights from training Gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[64] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents. *arXiv preprint arXiv:2204.06125*, 2022.

[65] Aqib Rashid and Jose Such. Malprotect: Stateful defense against adversarial query attacks in ml-based malware detection. *arXiv preprint arXiv:2302.10739*, 2023.

[66] Jessica Rumbelow and Matthew Watkins. SolidGoldMagikarp III: Glitch token archaeology. https://www.lesswrong.com/posts/8viQEp8KBg2QSW4Yc, 2023.

[67] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. LAION-5B: An open large-scale dataset for training next generation image-text models. *NeurIPS*, 2022.

[68] Roei Schuster, Jin Peng Zhou, Paul Grubbs, Thorsten Eisenhofer, and Nicolas Papernot. Learned systems security. *arXiv preprint arXiv:2212.10318*, 2022.

[69] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL*, 2016.

[70] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *NeurIPS*, 2018.

[71] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE S&P*, 2017.

[72] Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Diffusion art or digital forgery? investigating data replication in diffusion models. In *CVPR*, 2023.

[73] Liwei Song, Reza Shokri, and Prateek Mittal. Privacy risks of securing machine learning models against adversarial examples. In *ACM SIGSAC*, 2019.

[74] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2013.

[75] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[76] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX*, 2016.

[77] Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. Truth serum: Poisoning machine learning models to reveal their secrets. In *ACM SIGSAC*, 2022.

[78] Serge Vaudenay. Security flaws induced by CBC padding—applications to SSL, IPSEC, WTLS... In *Theory and Applications of Cryptographic Techniques*, 2002.

[79] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *IEEE/IFIP DSN*, 2020.

[80] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *ACSAC*, 2018.

[81] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open DNN box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[82] Qixue Xiao, Yufei Chen, Chao Shen, Yu Chen, and Kang Li. Seeing is not believing: Camouflage attacks on image scaling algorithms. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 443–460, Santa Clara, CA, 8 2019. USENIX Association. ISBN 978-1-939133-06-9. URL https://www.usenix.org/conference/usenixsecurity19/presentation/xiao.

[83] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/yan.

[84] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *IEEE CSF*, 2018.

[85] Zhanyuan Zhang, Yizheng Chen, and David Wagner. SEAT: similarity encoder by adversarial training for detecting model extraction attack queries. In *ACM Workshop on Artificial Intelligence and Security*, 2021.