A Benchmark for Training Data Extraction Attacks on Language Models

Nicholas Carlini, Christopher A. Choquette-Choo, Daphne Ippolito,
Matthew Jagielski, Katherine Lee, Milad Nasr, Florian Tramer and Chiyuan Zhang

Abstract

Recent work has demonstrated the feasibility of training data extraction attacks, where an adversary can interact with a pretrained language model to recover individual training examples contained in the training dataset. In this paper we develop a benchmark dataset and suite of metrics to evaluate the effectiveness of extraction attacks. We hope our benchmark will inspire future work to develop improved techniques. Our benchmark is available at `http://github.com/google-research/lm-extraction-benchmark`.

1 Introduction

Neural language models trained on sensitive datasets have been shown to memorize their training datasets, and with an *extraction attack* an adversary can recover individual training examples from the model's training dataset [Carlini et al., 2021]. However, recent work has found that there is a significant gap between what a model memorizes, and what current attacks can actually extract [Carlini et al., 2022, Kandpal et al., 2022].

In part we believe this gap can be explained by the difficulty in even evaluating the effectiveness of training data extraction attacks, making it difficult to develop better attacks. Even asking the basic question "was this string contained in a model's training dataset" is challenging on terabyte-sized datasets [Lee et al., 2021].

This paper develops a benchmark to evaluate training data extraction attacks. We focus on the problem of evaluating the "targeted" extraction attacks, where the adversary is provided with a prefix sequence and must find the specific continuation (suffix) such that the entire sequence is contained in the training dataset [Carlini et al., 2022]. For example, if the training dataset contains the sentence "My phone number is 123-4567", then the adversary may be given the prefix "My phone number is" and be asked to extract the suffix "123-4567" from a trained model with such sentence in its training set. This differs from the "untargeted" attacks, where an adversary searches for *any* memorized sample, i.e., any data that appears *somewhere* in the training dataset [Carlini et al., 2021].

2 The Benchmark

Dataset Our benchmark consists of a collection of 100-token sequences x_i from the Pile [Gao et al., 2020], which is a publicly available large language model dataset. Each of these examples is split into a 50-token prefix p_i and a 50-token suffix q_i, i.e. $x_i = p_i || q_i$. The adversary's objective is to guess q_i given each p_i and the query access to the GPT-Neo (1.3B) language model [Black et al., 2021], which is trained on the Pile.

The dataset is split into a training set of 15,000 examples, a validation set of 1,000 examples and a test set of 1,000 examples. The examples are randomly sampled from the Pile training set with the criterion that they are *not trivial* (e.g. not repeats of the same token), *extractable* (there is a prefix p_i' such that the GPT-Neo model output q_i given p_i'), and *well specified* (given p_i, the suffix q_i is unique in the *entire* Pile training set).

Evaluation We allow the adversary to make multiple guess for each extraction, and measures the fraction of examples extracted (i.e. the *recall*) under a precision constraint of no more than 100 total errors. For the accompanying challenge[1], we further enforce a computation constraint of roughly 24 hours of running time on a machine with a P100 GPU and 8 CPU cores. This constraint is mostly to make the evaluation easy, and we expect most attacks to use much less computation. Furthermore, it is easy to reconfigure the benchmark dataset with different evaluation criterion (see Section 3.5).

3 Designing the Benchmark

Our paper develops a benchmark to measure the efficacy of training data extraction attacks on language models. We begin by constructing a dataset D of sequences that are contained in the model's training dataset. We then run the attack by providing them with some limited information about the sequences in D, running the (randomized) attack algorithm $\mathscr{A}$ to obtain a set of training data guesses $\hat{X} = \{\hat{x}_i\}$. Finally, we evaluate the attacks using various recall-based metrics $M(D, \hat{X})$ that measure

[1]`https://github.com/google-research/lm-extraction-benchmark`.

(something like) $\frac{D \cap \hat{X}}{|\hat{X}|}$. Stronger adversaries will be able to cover more examples from D using less time, or by making fewer guesses $|\hat{X}|$.

3.1 Targeted Extraction Attacks

If we were to naively follow in the path of prior work we might decide to construct a benchmark that measures what fraction of some training dataset can be extracted by querying a pretrained language model. This would closely mirror what prior attacks aimed to do: given query access to a language model f, these attacks aim to identify some—indeed, any—data contained in the model's training dataset. While these *untargeted* attacks serve as a useful proof-of-concept that extraction attacks are feasible, they have two limitations that make them poorly suited to serve as a benchmark.

First, succeeding at this task encourages attacks that we would find uninteresting. Any sufficiently large collection of text is likely to contain uninteresting "common" texts, such as lists of number sequences, the names of countries in alphabetical order, sequences of the same character repeated hundreds of times, etc. Untargeted attacks would have the option to reach very high extraction rates just by guessing at thousands of such sequences. This encourages attacks to focus on techniques to generate benign and likely sequences as opposed to attacks that are designed to recover more interesting sequences from the training dataset.

The second reason we avoid untargeted attacks is they are less obviously privacy-relevant when compared to targeted attacks. For example, an adversary might want to be able to learn specific information about an individual user contained in the dataset. An untargeted attack can not do this.

Interesting subset. The above argument rules out choosing the entire training dataset $\mathscr{D}$ as our evaluation dataset. The next obvious choice would then be to take the dataset D, somehow remove the "uninteresting" sequences from D, and then measure recall as the adversary's ability to extract the "interesting" set of sequences (but without providing the adversary any information about this subset of the dataset).

While improved, this approach again has significant flaws. The first is that the task is not well specified: the adversary might very well be very successful at extracting some (other) training data, but if (for any reason) this data was deemed "uninteresting" and so not present in the dataset the adversary will have failed. But the adversary would have no way of actually knowing this. This results in a situation where we are in a sense asking researchers to "read our mind" about what kinds of training data we thought were interesting. The situation is further complicated by the fact that, once we release this benchmark, researchers will be able to overfit to the data we decided was interesting.

Targeted evaluations. We therefore instead choose an evaluation protocol that does first search for an interesting subset of the training data, but then to make the task well specified we give the adversary some additional information.

In particular, we split each sequence x contained in the training dataset into a prefix p and suffix q, which we denote by $x = p||q$. Then, we give the adversary a dataset of prefixes $\{p_i\}$ and ask the attack to produce the suffixes $\{q_i\}$. As long as we are careful in how we choose the sequences x, this ensures that the adversary's task is well-specified, and there now exists a single unique answer that can be recovered.

3.2 Design Criteria

We begin with a set of design criteria we would like our benchmark to satisfy.

Efficient evaluation. Evaluating the success rate of an untargeted attack is computationally expensive. After an adversary who proposes a set s of potentially-memorized sequences, to verify the success rate we must then check for the presence or absence of each sequence s_i in a large (e.g., terabyte-sized) dataset. While there do exist relatively efficient techniques to perform this lookup [Lee et al., 2021], even storing the 4TB suffix array would be challenging for many academic researchers. In contrast, our benchmark can be stored in a few megabytes and an evaluation takes a few seconds.

Cost-aware metric. Any evaluation methodology we use for extraction attacks must take into account the cost of the attack. In the limit, a computationally unbounded adversary could enumerate all length-k sequences in $O(n^k)$ time and achieve "perfect" recall at identifying all sequences in the training dataset. Such a solution is uninteresting due to its cost, both in terms of compute time (no adversary could ever instantiate this attack in practice) but also in false positives (because almost all generated sequences are not training data).

Any evaluation metric should therefore quantify the attack efficiency in some way. There are various strategies that could work here; for example, we could plot curves that compare the number of sequence extracted as a function of runtime—as is standard in the fuzzing literature. Alternatively, we could leave runtime unbounded but instead measure some precision-recall curve or true positive rate vs. false positive rate curve that compares how often the guesses made by an adversary are correct—as is done in membership inference attacks.

Only a few configurations. Even with the above desiderata, there are still a number of possible benchmarks that would be useful. While in principle it would be possible to, for every choice of parameters, create a benchmark tailored to those parameters, unfortunately this would result in hundreds of different benchmarks—and while it might be useful to be able to measure an attack specifically on exactly one configuration, this makes it difficult (or impossible) to compare related work.

Instead, we choose to design just three evaluation conditions that cover the design space, and deign our "main contest" around just one. We hope that this will more easily allow researchers to develop techniques and evaluate using the same set of benchmarks.

3.3 Evaluation Overview

We now present a brief overview of our approach. We begin by selecting a subset of examples $S = \{s_i\}$ from The Pile that have the property if we split s_i into a prefix p_i^* and suffix q_i then $f(p_i^*) = q_i$. That is, the model "knows" the right way to continue the example s_i. We then take a suffix of p_i^* as the *challenge prompts p_i*.

The adversary receives as input these partial prompts, and must output predictions for what the various values of q_i are. To do this, the adversary outputs a sorted list of guesses $g_j = (i'_j, q'_j)$. While they differ in their details, the general strategy we adopt to evaluate the effectiveness of an attack is by evaluating its "recall" under a set of either time-constraints or precision-constraints (or both).

1. When constrained by time, the adversary aims to maximize their recall in as short a time as possible.
2. When constrained by precision, the adversary aims to achieve a high recall while making as few incorrect guesses as possible (i.e., minimizing the false positive rate)
3. When constrained by both, the adversary must efficiently extract training examples without making too many errors. This is our "main challenge".

3.4 Our Dataset

We now provide additional details for how our dataset is constructed. At a high level, this dataset is designed to contain examples where the memoriation problem is both well-specified, nontrivial, and not impossible.

We call an example *well specified* intuitively if, given the prefix, there is only one "correct" answer for what should follow. For example, consider a prefix p where there were two possible continuations in the training dataset q and q'. If we selected the example $p||q$ (or $p||q'$) for our dataset, an adversary could "succeed" at extraction by recovering q' (respectively, q) even though this is the "wrong" answer according to our dataset. We want to avoid this class of errors, and so only include examples where the suffix is unique given the prefix.

An sequence is *nontrivial* if it's not immediately obvious what the continuation is. To identify trivial solutions we filter out sequences where the suffix has low lexical diversity (i.e., repeats the same token many times) or is repetitive (repeats the same sequence of tokens many times).

Finally, a sequence is *not impossible* if there exists some plausible strategy that the adversary could use to recover q. A first prerequesite of this we require that there exists a prefix p such that $f(p) = q$—this enforces the constraint that the model does, in fact, "know" the solution q [cite extraction]. We additionally place the requirement that q is not a substring of p. Otherwise it is possible that we could encounter a sequence of the form $n||b||n$ for some nonce n. This sequence would be considered possible because there does exist a prefix such that the model outputs n, but this sequence is uninteresting because in practice we've told the model what the answer should be.

3.4.1 Downselection process

Exactly which sequences $s = p||q$ we choose to allow in our dataset matters. We remove various sequences from the dataset if any of the following properties apply.

Unique sequences. Our dataset is designed to maximize the statistical power of the evaluation while minimizing the computational resources necessary to evaluate the attack. While it would be maximally informative to use the entire 800+GB training dataset as our benchmark and attempt attack on every sequence one by one, this would be too slow.

Therefore, as a first filter, we down-select the 800GB pile dataset to under 10GB of training data that has a possibility of being memorized. Given that The Pile is 800+GB, and the largest GPT-Neo model is just 20GB, it is unlikely that the model could have memorized the entire dataset. Indeed, prior work has found that language models memorize only a few percent of their datasets. Recent work has found that duplicate data [Carlini et al., 2022, Kandpal et al., 2022] is one of the primary factors for why training data gets memorized. Therefore, as a first pass, we only consider sequences that are repeated at least 5 times somewhere in The Pile. While it is possible that we might miss some sequences that are memorized with this initial first pass, it significantly speeds up our construction.

Underspecified sequences. Because we will be making a binary success/failure decision for each sequence based on whether or not $\mathscr{A}(p) = q$, we must ensure that q is the only valid answer given the prefix p. However there are often cases where one given prefix p is contained in the dataset twice, where in one instance it is followed by q_1 and in another it is followed by q_2. (That is, $p||q_1 \in \mathscr{D}$ and $p||q_2 \in \mathscr{D}$.) Therefore, asking the question "what comes next after p?" is not a well defined question—both q_1 and q_2 do. To resolve this ambiguity we remove any sequences from the dataset where there is not a single unique continuation.

Trivial sequences or self-referential sequences.. Some sequences, such as those that repeat the same character hundreds of times in a row, are uninteresting from an attack perspective. We remove these sequences by filtering out any sequence with an entropy below 1-bit-per-token.

3.5 Evaluation Procedure

We evaluate extraction attacks that are both time- and precision-constrained

3.5.1 Time-based Evaluation

We begin by describing an evaluation strategy that mirrors how fuzzing works, and focuses exclusively on evaluation run-time. We assume there exists an oracle which answers the question "is this sequence contained in the training dataset?" (analogous to the fuzzing oracle that answers "does this program crash?"). The objective of the algorithm is to identify as many sequences contained in the dataset, with as few errors, in as little time, as possible.

How do we measure "time"? Unlike in fuzzing, there can be factor-of-a-thousand performance differences between various hardware configurations, and so comparing based directly on runtime may not be desirable. To fix this, we specify a specific hardware that should be used to compare evaluations across time—and if necessary a scaling factor could be used in the future to compare to current work.

3.5.2 Precision-based Evaluation

Recall the adversary in our setting outputs an ordered sequence of examples (i'_j, q'_j) and our evaluation metric Recall at K.

The one evaluation criterion that is not considered when evaluating in this way is the computational complexity of attacks. Imagine that we had some membership inference attack that was perfect, that is, it had a precision of 100% at a recall of 100%. A "solution" to our challenge would then, again, be to enumerate all potential sequences and then choose those where the membership inference attack succeeds. More realistically, given a membership inference attack with high true positive rate at a low false positive rate, an adversary who generates 1 million candidate sequences and orders them with the membership inference attack might lose out to the adversary who enumerates 10 million sequences, who might yet still lose out to the adversary who first considers a 1 billion sequences.

3.5.3 Time- and precision–based Evaluation

We choose to evaluate using both constraints simultaneously. That is, our adversary is bounded both by time and yet also aims to develop high-precision attacks.

References

[Black et al., 2021] Black, S., Gao, L., Wang, P., Leahy, C., and Biderman, S. (2021). Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow. *If you use this software, please cite it using these metadata*, 58.

[Carlini et al., 2022] Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. (2022). Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*.

[Carlini et al., 2021] Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. (2021). Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650.

[Gao et al., 2020] Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

[Kandpal et al., 2022] Kandpal, N., Wallace, E., and Raffel, C. (2022). Deduplicating training data mitigates privacy risks in language models. *arXiv preprint arXiv:2202.06539*.

[Lee et al., 2021] Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. (2021). Deduplicating training data makes language models better. In *The 60th Annual Meeting of the Association for Computational Linguistics*.