Bag of Tricks for Training Data Extraction from Language Models

Weichen Yu [*] [1] Tianyu Pang [2] Qian Liu [2] Chao Du [2] Bingyi Kang [2] Yan Huang [1] Min Lin [2] Shuicheng Yan [2]

Abstract

With the advance of language models, privacy protection is receiving more attention. Training data extraction is therefore of great importance, as it can serve as a potential tool to assess privacy leakage. However, due to the difficulty of this task, most of the existing methods are proof-of-concept and still not effective enough. In this paper, we investigate and benchmark tricks for improving training data extraction using a publicly available dataset. Because most existing extraction methods use a pipeline of generating-then-ranking, i.e., generating text candidates as potential training data and then ranking them based on specific criteria, our research focuses on the tricks for both text generation (e.g., sampling strategy) and text ranking (e.g., token-level criteria). The experimental results show that several previously overlooked tricks can be crucial to the success of training data extraction. Based on the GPT-Neo 1.3B evaluation results, our proposed tricks outperform the baseline by a large margin in most cases, providing a much stronger baseline for future research. The code is available at https://github.com/weichen-yu/LM-Extraction.

1. Introduction

Recent advances in language models (LMs) have led to impressive performance in a variety of downstream language tasks (Kenton & Toutanova, 2019; Brown et al., 2020). It has been demonstrated, however, that training data can be extracted from LMs due to the memorization effects (Kenton & Toutanova, 2019; Carlini et al., 2019; Feldman, 2020; Brown et al., 2020). These training data may contain sensitive information such as names, email addresses, phone numbers, and physical addresses, resulting in privacy leakage that hinders the widespread adoption of LMs (Carlini et al., 2021; 2022; Lehman et al., 2021).

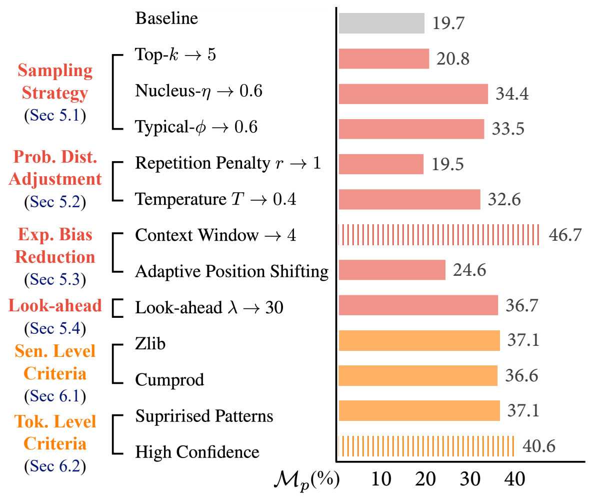

Figure 1. Overview for the bag of tricks explored in this work, with an evaluation of precision ($\mathcal{M}_P$). Bars in pink denote the methods in the improved suffix generation, and bars in orange denote the methods in the improved suffix ranking. The *dashed bars* indicate the best method in each category.

As privacy security has been an important issue of public concern, a crucial topic is to develop efficient methods for evaluating privacy leakage. Thus, the focus of our research is on the adversarial task of training data extraction from LMs, a relatively new area of study (Carlini et al., 2021; Lehman et al., 2021). Existing extraction methods have yielded successful records, but there are instances in which these methods are even less effective than simply selecting the most popular entity based on prior score. In addition, successful data extraction requires a high generation ratio, i.e., the need to generate and rank a large number of candidates in order to identify a single successful instance. These suboptimal results suggest that, despite the viability of training data extraction and developed pioneering methods as demonstrated in previous research, this task is still relatively new with an abundance of problems to solve.

In this study, we aim to develop techniques for efficient training data extraction. We adhere to the criteria of the recent Training Data Extraction Challenge,[1] which employs

[*]Work done during an internship at Sea AI Lab. [1]Institute of Automation, Chinese Academy of Sciences. [2]Sea AI Lab. Correspondence to: Tianyu Pang <tianyupang@sea.com>, Qian Liu <liuqian@sea.com>, Yan Huang <yhuang@nlpr.ia.ac.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]Website link of Training Data Extraction Challenge

a 1.3B parameter GPT-Neo model (Black et al., 2021) for *targeted extraction of* 1*-eidetic memorized data*. Targeted extraction refers to the scenario where a prefix of the data is provided, such as 'Yu's phone number is', and the adversary attempts to recover the suffix '12345'. Accroding to Carlini et al. (2021), κ-eidetic memorization is defined as the capacity of a language model to memorize a string that appears κ times in the training material. The targeted and 1-eidetic extraction poses a greater risk and is more challenging than non-targeted and κ-eidetic (for $\kappa > 1$) settings.

Through ablation studies, we assess a variety of simple techniques in natural language processing (NLP) and empirically evaluate their impact on successful extraction rates, as in Figure 1. Our empirical analysis reveals that the extraction performance may be sensitive to the experimental setup. With proper settings, the results show that several previously overlooked tricks can contribute to significant improvements of training data extraction. Based on the GPT-Neo 1.3B evaluation results, our proposed tricks outperform the baseline by a large margin in most cases, providing a much stronger baseline for future research. Nonetheless, utilizing more than one training data extraction trick does not necessarily boost the performance, and in some cases, even shows incompatibility and results in inferior precision. These findings suggest that judicious selection and combination of the tricks are essential for optimal performance.

2. Related Work

We briefly introduce training data extraction, membership inference attacks, and other memorization-based attacks.

2.1. Training Data Extraction

The extraction of training data from a pretrained language model, also referred to as language model data extraction, is a method for recovering the examples used to train the model. Despite being a relatively new task, many of the underlying technologies and analysis methods, including membership inference (Shokri et al., 2017) and leveraging network memorization for attacks (Thomas et al., 2020; Leino & Fredrikson, 2020), were introduced much earlier.

Carlini et al. (2021) were among the first to define the concepts of model knowledge extraction and κ-eidetic memorization, as well as to propose promising training strategies for data extraction. Both the theoretical properties of memorization and the application of model extraction in sensitive fields, such as the analysis of clinical notes, have been the focus of subsequent research in this field.

Recently, Kandpal et al. (2022) demonstrated that in language models, the efficacy of data extraction is frequently attributable to duplication in commonly used web-scraped training sets. Using nondeterminism, Jagielski et al. (2022) provided an explanation for forgetting memorized examples. Carlini et al. (2022) analyzed three factors that affect the memorization of training data. For natural data distributions, Feldman (2020) showed that label memorization is required to achieve near-optimal performance. In the application of model extraction, Lehman et al. (2021) indicated that pretrained BERT, when trained on clinical notes, poses a risk of sensitive data leakage, especially when the data exhibits a high level of repetition or 'note bloat' (Liu et al., 2022). Jayaraman et al. (2022) proposed an active extraction attack that employs canonical patterns and differential privacy to defend against pattern extraction attacks.

2.2. Membership Inference Attacks

The membership inference attack (MIA) (Shokri et al., 2017) is another adversarial task in data protection that is closely associated with training data extraction. It aims to determine whether a particular record is present in its training dataset, given black-box access to a model. MIA has been demonstrated to be effective on numerous machine learning tasks, including classification (Sablayrolles et al., 2019; Choquette-Choo et al., 2021; Rezaei & Liu, 2021) and generation (Hayes et al., 2019; Hilprecht et al., 2019).

The methods utilized by MIA fall into two categories: classifier-based methods and metric-based methods (Hu et al., 2022). Classifier-based methods involve training a binary classifier to recognize the complex relationship between members and non-members, with shadow training being a commonly used technique (Shokri et al., 2017; He et al., 2020; Wang et al., 2021). Metric-based methods, on the other hand, make membership inferences by first calculating metrics on the model prediction vectors (Yeom et al., 2018; Salem et al., 2018; Sablayrolles et al., 2019; Song & Mittal, 2021; Choquette-Choo et al., 2021). Several defense methods based on differential privacy (Leino & Fredrikson, 2020; Naseri et al., 2020; Choquette-Choo et al., 2021), data pruning (Wang et al., 2021), data augmentation (Kaya & Dumitras, 2021) and causal inference (Tople et al., 2020) have been proposed to mitigate this vulnerability.

2.3. Other Memorization-Based Attacks

It has been discovered that large pretrained models are susceptible to memorizing information from the training data, which can lead to a variety of attacks. In addition to training data extraction and membership inference attacks, there are other memorization-based attacks that target these models.

Model extraction attacks and the corresponding protection methods (Tramèr et al., 2016; Juuti et al., 2019; Gong et al., 2020; Wu et al., 2022) focus on the issue of duplicating the functionality of a given model. In this type of attacks, the adversary attempts to build a second model with a similar predictive performance to the original black-box model.

The objective of attribute inference attacks is to extract specific personal attributes such as locations, occupations, and interests from the model (Fredrikson et al., 2015; Gong & Liu, 2016; Ganju et al., 2018; Parisot et al., 2021). The objective of property inference attacks is to extract properties of the training data that the model producer may not have intended to share, such as the environment in which the data was generated or the proportion of the data that belongs to a particular class. The primary distinction between training data extraction attacks and attribute/property inference attacks is that attribute/property inference attacks do not require prior knowledge of the attributes or properties to be extracted, whereas training data extraction attacks require the generated information to be identical to the training data at the sentence level, which is more difficult and dangerous.

3. Preliminary

We recap the basic setups employed in our study. These setups mainly follow the guidelines of the Training Data Extraction Challenge. We then define the threat model and evaluation metrics.

3.1. Basic Setups

Dataset. The dataset used in this study is a subset of 20,000 examples from the Pile's training dataset (Gao et al., 2020). Each example consists of a 50-token prefix and a 50-token suffix. The attacker's task is to predict the suffix given the prefix. All the 100-token long sentences in this dataset appear only once in the training set. For the purposes of this study, we divide the dataset into a training set of 19,000 samples and a testing set of 1,000 samples.

Language model. We employ the GPT-Neo 1.3B model implemented on HuggingFace Transformers (Wolf et al., 2020), which is a transformer model designed using EleutherAI's replication of the GPT-3 architecture (Brown et al., 2020), and trained on the Pile dataset. GPT-Neo is an autoregressive language model f_θ parameterized by θ, which generates a sequence of tokens $x_0, x_1, \cdots, x_N$ via the chain rule

$$f_\theta\left(x_0, x_1, \cdots, x_N\right) = \prod_{n=0}^{N} f_\theta\left(x_n | x_{[0,n-1]}\right), \qquad (1)$$

where $x_{[0,n-1]} = x_{<n} = \{x_0, \cdots, x_{n-1}\}$ for notation compactness. At the sentence level, given a prefix p, we denote the probability of generating a certain suffix s conditional on the prefix p as $f_\theta(s|p)$.

3.2. Threat Model

In this study, we focus on the threat model of targeted extraction of κ-eidetic memorized data, where we choose $\kappa = 1$. According to the model knowledge extraction defined in (Carlini et al., 2021), we assume the language model generates a suffix s by the most-likely criterion. Then we can write a formal definition of targeted extraction as

Definition 1. *(Targeted extraction) Given a prefix p contained in the training data and a pretrained language model f_θ. Targeted extraction is to generate the suffix by $s = argmax_{s'} f_\theta(s'|p)$.*

As to κ-eidetic memorized data, we follow the definition in Carlini et al. (2021) that the sentence $[p, s]$ appears in at most κ examples in the training data. In practice, the length of the generated sentence is typically fixed using truncating and concatenation techniques applied to the training dataset. If a generated sentence is shorter than the specified length, padding tokens are used to bring it up to the required length. In this study, the generated sentence length is 100.

3.3. Evaluation Metrics

Non-targeted extraction has been evaluated using the number of memorized examples in previous studies (Carlini et al., 2021). To evaluate more comprehensively, we use three metrics to evaluate performance in this targeted data extraction task, including precision $\mathcal{M}_\text{P}$, recall $\mathcal{M}_\text{R}$ and Hamming distance $\mathcal{M}_\text{H}$.

Precision $\mathcal{M}_\text{P}$. The proportion of correctly generated suffixes over the total number of given prefixes is referred to as precision $\mathcal{M}_\text{P}$. Notice that for a correct generation, the suffix and ground truth must be identical in both sentence length and generated tokens.

Recall $\mathcal{M}_\text{R}$. The proportion of correctly generated suffixes over the total number of generated suffixes is indicated by recall $\mathcal{M}_\text{R}$. The metric used in the Training Data Extraction Challenge is denoted by $\boldsymbol{e}_j$, which is defined as the number of correctly recovered suffixes when the number of incorrectly generated suffixes reaches a threshold of j. $\boldsymbol{e}_j$ can assess the effectiveness of the attack. We define $\mathcal{M}_\text{R} = \frac{\boldsymbol{e}_j}{\boldsymbol{e}_j + j}$, we will use $\mathcal{M}_\text{R}$ instead of e_j in the following paragraphs. In our experiments, the value of j is chosen to be proportional to the size of the test set, and it is set to 100 with a test set of 1,000 prefixes.

Hamming distance $\mathcal{M}_\text{H}$. The Hamming distance denotes the difference between two equal-length strings, calculated as the number of positions where the corresponding symbols differ. We can quantitatively evaluate the similarity between the generated suffixes and the ground truth using the average Hamming distance, providing a token-level evaluation of the extraction methods' performance. $\mathcal{M}_\text{H} = \frac{1}{N} \sum_n x_n \oplus gt_n$, where $a \oplus b = 1$ if $a = b$, else 0. N is the number of tokens in a generated sentence, x_n is the generated token, and gt_n is the corresponding ground truth token.

We would like to point out that the commonly used metrics that measure the repetition or quality of generated sentences, e.g., the Zipfian coefficient and REP (Welleck et al., 2019), may not be aligned with the goal of training data extraction. Thus, we propose the metrics listed above, which are specifically designed to assess the efficacy of training data extraction methods. These metrics, $\mathcal{M}_{\mathrm{P}}$, $\mathcal{M}_{\mathrm{R}}$, and $\mathcal{M}_{\mathrm{H}}$, are closely linked with the success rate of the extracted data rather than the quality of generated language.

4. Pipeline Overview

A basic training data extraction pipeline can be divided into two stages. The first stage is *suffix generation*, which means coming up with a set of suffixes based on a prefix. The autoregressive language model f_θ computes the probability of each token in a vocabulary list and generates the next token by sampling from the probability distribution. A basic strategy employed to control the number of generations is limiting to the top-k tokens with $k = 10$, which means the LMs only sample from the tokens with top-k probability. The second stage is *suffix ranking*, which entails estimating the likelihood of these suffixes and retaining only those with a high probability. Typically this is accomplished through the use of membership inference attacks, determined based on the perplexity metric (Carlini et al., 2019; 2021) as

$$\mathcal{P} = \exp\left(-\frac{1}{N}\sum_{n=0}^{N}\log f_\theta(x_n|x_{[0:n-1]})\right), \qquad (2)$$

where N is the number of tokens in a generated sentence. Here, padding tokens are not included in the calculation. We explore and evaluate various tricks for both stages. The results for suffix generation are presented in Section 5, and the results for suffix ranking are presented in Section 6.

5. Improved Suffix Generation

To improve suffix generation, we analyze the logits distributions of both the ground-truth and generated tokens. As shown in Figure 2, there is a significant difference between the two distributions. To address this, we examine the effects of a variety of NLP techniques, such as changes to sampling strategies and probability distributions, as described below.

5.1. Sampling Strategy

The most popular decoding objective, especially for conditional generation in NLP tasks, is maximization-based decoding. Based on the provided prefix, it searches for the suffix with the highest likelihood. It is also suited for our training data extraction task since the models are directly maximized for the likelihood of the training data. However, finding the theoretically argmax sequence from models is intractable (Holtzman et al., 2019). The common practice is using beam search (Freitag & Al-Onaizan, 2017) instead.

Table 1. Results of $\mathcal{M}_{\mathrm{P}}$, $\mathcal{M}_{\mathrm{R}}$, and $\mathcal{M}_{\mathrm{H}}$ under different numbers of beams in beam search. All results are reported on a single trial.

Beam	**Memory**	$\mathcal{M}_{\mathrm{P}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{R}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{H}}\downarrow$
7	21531M	51.1	76.3	15.501
6	21379M	51.0	76.4	15.575
5	20731M	51.3	76.3	15.499
4	17051M	50.7	76.3	15.763
3	14333M	50.7	76.2	15.764
2	12023M	50.6	76.2	15.931
1	8783M	37.0	76.5	20.245

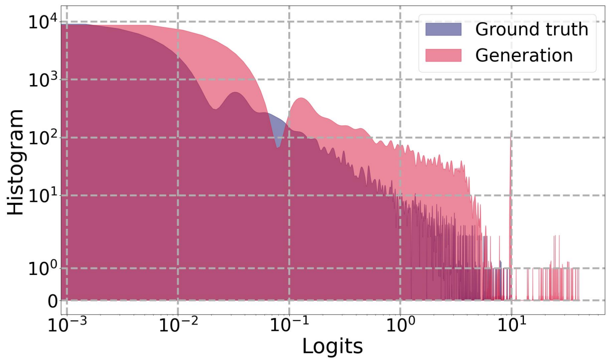

Figure 2. Histogram of token logits. The histogram depicts the distribution of logit values obtained from 1,000 suffixes, each containing 50 tokens. A spline interpolation technique is employed to smooth the histogram, with the original histogram included in the appendix for reference.

As an approximate solution, beam search only keeps a predetermined number of best partial solutions. Since it always takes the top ones at each step, it is often criticized for its lack of diversity (Holtzman et al., 2019). As shown in Table 1, although the required memory is increasing, the performance gain from a larger beam width diminishes quickly when the beam width exceeds two. Due to the randomness in generation, sampling methods always introduce more diversity than beam search. Therefore, we mainly study sampling strategies below, including top-k sampling, nucleus sampling and typical sampling.

Top-k denotes the truncated sampling method with the truncation set defined as the top-k highest-probability tokens (Fan et al., 2018; Holtzman et al., 2018; Radford et al., 2019). Figure 3(a) demonstrates that a larger k may increase the diversity of the generated sentences but also decrease the precision and hamming distance. The result is consistent with that observed in other NLP tasks Meister et al. (2022).

Nucleus-η (Holtzman et al., 2019) truncates the vocabulary based on the summed probability. Namely, the smallest set of the most likely tokens with total probabilities equal to or greater than η are selected. Meister et al. (2022) demonstrate that lower values of η are more conducive to story generation. In contrast, the extraction task is optimal at $\eta \approx 0.6$, which yields a 31% improvement in extraction precision over the baseline. Larger or smaller η both shows decreased performance (Figure 3(b)).

Typical-ϕ (Meister et al., 2022) recommends selecting a token with information content similar to the expected information content. Namely, it guarantees the probability mass from the original distribution that will be taken into account in typical decoding. This sampling strategy can improve sentence consistency while reducing degenerate repetitions. The typical-ϕ strategy is equivalent to a subset optimization problem with an entropy rate constraint. Typical-ϕ exhibits a non-monotonic tendency, which is similar to its effect on abstract summary and story generation tasks.

5.2. Probability Distribution Adjustment

Besides the sampling strategies that truncate the sampling distribution, we introduce in this section another strategy that directly adjusts the probability distribution $f_\theta(x_n|x_{<n})$. Below we introduce two tricks, adjusting the temperature and repetition penalty, to refine the distribution. These tricks bring nearly no additional computation cost but can still lead to a performance improvement.

Temperature T (Hinton et al., 2015) is a technique of local renormalization with an annealing term. A higher temperature $T > 1$ results in decreased confidence in the language model's predictions but may also increase the diversity of the generated suffixes. The study conducted by Carlini et al. (2021) found that gradually decreasing the temperature throughout the generation process can be beneficial. The effect of the temperature is presented in Table 2. It is important to note that as the temperature is increased, the number of generated suffixes required to include the ground truth also increases, causing the efficiency to degrade. It is important to find a balance between diversity and efficiency.

Repetition penalty is constructed on the conditional language model (Keskar et al., 2019). A repetition penalty is introduced by locally modifying the generation probability of each token based on whether it is a repetition of the previous token. The logit of the repeated token is divided by a value r before entering the softmax layer. Setting $r > 1$ penalizes repetition while $r < 1$ encourages it. Our results in Table 3 show that repetition penalty has mostly negative effects on the task of training data extraction.

5.3. Exposure Bias Reduction

For efficient vectorization, it is common practice to pack multiple sentences into a fixed-length sequence when training language models. As an example, consider the sentence 'The phone number of Yu is 12345' may be truncated or prefixed with another sentence in the training set, such as

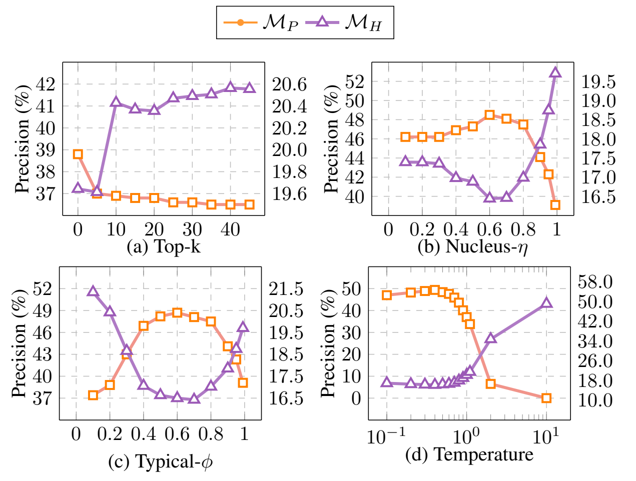

Figure 3. Experimental results under different values of top-k, nucleus-η, typical-ϕ and temperature T. All results are reported on 5 trials. The y-axis left denotes precision (%)(↑), and right denotes Hamming distance (↓).

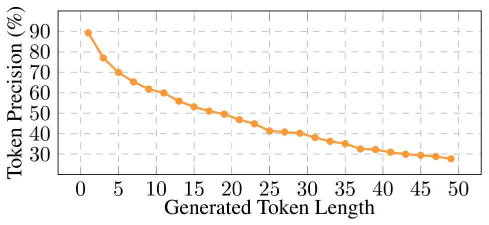

Figure 4. Generated token length w.r.t. token precision (%) for the n-th generated token. The generated suffix length is 50.

'number of Yu is 12345' or 'Yu's address is at XXX. The phone number of Yu is 12345'. The prefix in the training set, as in Table 12, is not always a complete sentence. To better mimic the training settings, we propose to adjust the context window size and adjust the position shifting.

5.3.1. Dynamic Context Window

The length of the training window may differ from the length of the extraction window. As a result, we propose adjusting the context window size, i.e. the number of previously generated tokens, as shown in Eq. (3). Furthermore, we encourage the results of different context window sizes to collaborate in determining the next generated token as

$$\begin{aligned} & f_\theta(x_n; \mathcal{W}) \\ & = h_{\mathcal{W}}\left(f_\theta\left(x_n|x_{[n-w_1,n-1]}\right), ..., f_\theta\left(x_n|x_{[n-w_m,n-1]}\right)\right), \end{aligned} \tag{3}$$

where $h_{\mathcal{W}}$ denotes the ensemble method, $\mathcal{W}$ denotes the ensemble hyperparameter, including the number of different context window sizes m and each window size w_i. We use $m = 4$ and $w_i \in \{n, n-1, n-2, n-3\}$ in our methods. Carefully chosen hyperparameters m and w_i may improve the performance even more.

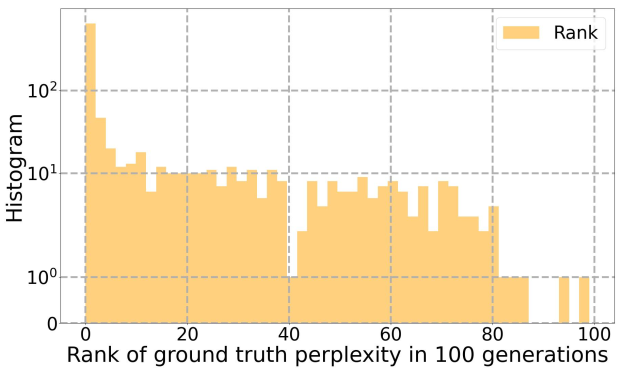

Figure 5. Histogram of the rank of the ground truth perplexity. The x-axis represents the rank of the ground truth perplexity within a list of 100 suffix perplexities.

We present two implementation options. The first option, as specified in Eq. (4), entails computing the probabilities generated by utilizing various lengths of previously generated tokens $x_{[n-w_i,n-1]}$, and then producing the final probabilities via a weighted average sum of these probabilities, as

$$f_\theta\left(x_n; \mathcal{W}_w\right) = \frac{\sum_{i=1}^{m} \epsilon_i f_\theta\left(x_n | x_{[n-w_i,n-1]}\right)}{\sum_{i=1}^{m} \epsilon_i}, \quad (4)$$

where $\mathcal{W}_w$ denotes the hyperparameters in the solution, comprising of m, w_i and ϵ_i. ϵ_i denotes the weighting coefficient of each probability. The second option as specified in Eq. (5), is based on a voting mechanism, in which each model trained with a distinct context window length casts its vote for the tokens it is most confident in, formulated as

$$f_\theta(x_n; \mathcal{W}_v) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{V}(f_\theta(x_n|x_{[n-w_i,n-1]}); \quad (5)$$

$$\mathcal{V}(f_\theta(x_n)) = \begin{cases} \rho - \mathcal{R}(f_\theta(x_n)), & \text{if } \mathcal{R}(f_\theta(x_n)) \leq \rho; \\ 0, & \text{otherwise,} \end{cases} \quad (6)$$

where $\mathcal{V}(\cdot)$ denotes the voting function, $\mathcal{R}(\cdot)$ denotes the rank function, and it votes for the tokens that it has confidence in. $\mathcal{W}_v$ denotes the hyperparameters in the solution, comprising of w_i, m and ρ.

It is stated in Carlini et al. (2022) that the proportion of extractable sequences increases log-linearly with the number of context tokens. We observed a similar phenomenon in our experiments, where the generation accuracy of a token decreases as the prefix becomes shorter. However, we discovered that combining multiple context window lengths significantly improves accuracy. The probabilities produced by different window lengths can be combined to significantly improve extraction accuracy. Our implementation of $\mathcal{W}_w$ employs the weighting coefficient $\epsilon_i = 0.9^i$, and $\mathcal{W}_v$ assigns $[5, 4, 3, 2, 1]$ points to its top-5 confident tokens, $\rho = 5$. The results presented in Table 4 show that

Table 2. Results of $\mathcal{M}_{\text{P}}$, $\mathcal{M}_{\text{R}}$, and $\mathcal{M}_{\text{H}}$ under different temperature. Temperature = 1 is the baseline. All results are reported on 5 trials.

Temperature	$\mathcal{M}_{\text{P}}$ (%)(↑)	$\mathcal{M}_{\text{R}}$ (%)(↑)	$\mathcal{M}_{\text{H}}$ (↓)
Varying	48.0	76.3	19.614
0.3	48.9	76.4	16.341
1	37.0	76.5	20.245

Table 3. Results of $\mathcal{M}_{\text{P}}$, $\mathcal{M}_{\text{R}}$, and $\mathcal{M}_{\text{H}}$ under different repetition penalty. Repetition penalty $r = 1$ is the baseline. All results are reported on 5 trials.

Repetition penalty	$\mathcal{M}_{\text{P}}$ (%)(↑)	$\mathcal{M}_{\text{R}}$ (%)(↑)	$\mathcal{M}_{\text{H}}$ (↓)
0.9	19.8	66.4	27.927
1	37.0	76.5	19.614
1.1	37.3	76.5	20.181
1.2	37.1	76.5	20.323
1.3	36.7	76.4	20.332
1.5	34.7	75.7	21.154

ensemble methods can significantly improve on the baseline approach, achieving improvements of 143% and 139%, respectively, and we discovered that the weighted average strategy performs slightly better than the voting strategy. One common failure mode observed is that when a wrong token is generated, it causes subsequent tokens to also be wrong (exemplars shown in Table 12). The window size ensemble introduced here can help reduce this problem.

5.3.2. Dynamic Position Shifting

Positional embeddings are added to the token feature in models like GPT-Neo. During training, this is added per batch of sentences, causing the same sentence to have different offsets in positional embedding in different training batches and during generation. To improve the extraction of memorized suffixes, we propose to recover the positions used during training by evaluating different shifted positions and selecting the 'best' one. Specifically, for a given prefix p, we evaluate different position $\mathcal{C} = c^i$, where c^i is a list of consecutive natural numbers, $c^i = \{c_1^i, \cdots\}$, s.t. $|c^i| = |p|$ and calculate the corresponding perplexity values. The position with the lowest perplexity value is then chosen as the position from which to generate the suffix as

$$c = \operatorname*{argmin}_{c^i \in \mathcal{C}} \mathcal{P}(p, c^i); \quad \hat{\phi}(x_i) = \psi(c_n) + \phi(x_n), \quad (7)$$

where $\psi(\cdot)$ denotes positional encoding layers, $\phi(\cdot)$ denotes the feature mapping function, $\hat{\phi}$ denotes the feature mapping function consisting positional encoding, and $\mathcal{P}$ computes the perplexity of the prefix. The experimental results are presented in Table 4. $\mathcal{C} = \{[0, 1, \cdots, |p|], [1, 2, \cdots, |p| + 1], \cdots\}$ is evaluated. The data show that, while using posi-

Table 4. Results of $\mathcal{M}_\text{P}$, $\mathcal{M}_\text{R}$, and $\mathcal{M}_\text{H}$ under context window length adjustments. All results are reported on a single trial.

	$\mathcal{M}_\text{P}$ (%)(↑)	$\mathcal{M}_\text{R}$ (%)(↑)	$\mathcal{M}_\text{H}$ (↓)
Baseline	19.5	65.6	26.948
Context Win $\mathcal{W}_w$	47.4	77.6	16.993
Context Win $\mathcal{W}_v$	46.7	77.5	17.164
Position Shifting	16.4	39.0	21.154

Table 5. Results of $\mathcal{M}_\text{P}$, $\mathcal{M}_\text{R}$, and $\mathcal{M}_\text{H}$ under auto-tuning. All results are reported on 5 trials.

Strategy	$\mathcal{M}_\text{P}$ (%)(↑)	$\mathcal{M}_\text{R}$ (%)(↑)	$\mathcal{M}_\text{H}$(↓)
Baseline	37.0	76.5	19.614
Manual selection	48.8	76.4	16.379
Auto-tuning	49.4	76.6	16.127

tion shifting improves the $\mathcal{M}_\text{H}$ metric, it may have a negative impact on precision and recall.

5.4. Look-Ahead

Table 12 highlights a common issue encountered during the training data extraction process, where only one or two tokens are incorrectly generated or placed in an inappropriate position. To address this problem, we propose a technique that involves looking ν steps ahead and using the probability of the subsequent tokens to inform the generation of the current token. The goal of look-ahead is to use the posterior distribution to help compute the current token generation probability. We begin by presenting the precise mathematical formulation of the optimal probability and then introduce the implementation, which employs an estimation due to efficiency considerations. The posterior is calculated as

$$\begin{aligned} &f_\theta\left(x_n|x_{n+1},x_{<n}\right)\\ =&\frac{f_\theta\left(x_{n+1}|x_n,x_{<n}\right)f_\theta\left(x_n|x_{<n}\right)}{\sum_{x'_n}f_\theta\left(x_{n+1}|x'_n,x_{<n}\right)f_\theta\left(x'_n|x_{<n}\right)}.\end{aligned} \tag{8}$$

More generally, let $\textbf{Track}(x_\text{start},x_\text{end}|x_\text{cond})$ be the probability product of the track starting from x_start and ending at x_end, conditioned on x_cond. Then we can write ν-step posterior as

$$f_\theta\left(x_n|x_{n+\nu},x_{<n}\right)=\frac{\textbf{Track}(x_n,x_{n+\nu}|x_{<n})}{\sum_{x'_n}\textbf{Track}(x'_n,x_{n+\nu}|x_{<n})}, \tag{9}$$

where **Track** is calculated as,

$$\begin{aligned}&\textbf{Track}(x_\text{start},x_\text{end}|x_\text{cond})\\ &=\sum_{x'_{\text{start}+1}}\sum_{x'_{\text{start}+2}}\cdots\sum_{x'_{\text{end}-1}}f_\theta(x_\text{start}|x_\text{cond})\\ &\qquad f_\theta(x_{\text{start}+1}|x_\text{cond},x_\text{start})\cdots\\ &\qquad f_\theta(x_\text{end}|x_\text{cond},x_\text{start},x'_{\text{start}+1}\cdots,x'_{\text{end}-1}).\end{aligned} \tag{10}$$

Table 6. Hyper-parameters selection for auto-tuning. Multiple configurations of final hyperparameters are found to yield equivalent performances, and a representative example is presented.

Parameters	**Range**	**Step**	**Initial**	**Final**
Top-k	[1, 50]	1	10	24
Nucleus-η	[0.1, 1]	0.01	0.6	0.8
Typical-ϕ	[0.1, 1]	0.01	0.6	0.9
Temperature T	[0.1, 5]	0.1	0.3	0.58
Repetition Penalty	[0.8, 1.3]	0.01	1	1.04

Table 7. Experimental results of $\mathcal{M}_\text{P}$, $\mathcal{M}_\text{R}$, and $\mathcal{M}_\text{H}$ under look-ahead. All results are reported on a single trial.

		$\mathcal{M}_\text{P}$(%)(↑)	$\mathcal{M}_\text{R}$(%)(↑)	$\mathcal{M}_\text{H}$ (↓)
Baseline		19.5	65.6	26.948
	λ=10	33.1	71.6	24.262
Look-ahead	λ=20	35.5	72.6	22.157
	λ=30	36.7	73.0	21.333

Based on the 1-step look-ahead formulation in Eq. (8), we utilize the posterior probability to calculate the confidence of the current token. However, due to the size of the vocabulary list (more than 50,000), we only select λ tokens whose probability $f_\theta\left(x'_n|x_{<n}\right)$ ranks among the highest, as

$$\mathcal{X}=\{x'_n|\mathcal{R}(f_\theta\left(x'_n|x_{<n}\right))\geq\lambda\}, \tag{11}$$

where $\mathcal{R}(\cdot)$ is the rank function. The experimental results, as detailed in Table 4, show that using a look-ahead strategy leads to a significant improvement in precision. Furthermore, an increase in the value of λ results in a corresponding improvement in performance. It is important to note, however, that increasing the value of λ also increases the computational cost, with a complexity of $\mathcal{O}(\lambda\cdot N)$, where N denotes the length of the generated tokens.

5.5. Hyperparameter Optimization

The aforementioned tricks in Section 5.1 involve various hyperparameters, and simply using the best parameters is usually suboptimal. Manually searching for the best hyperparameters, also known as 'babysitting,' can be time-consuming. We use a versatile architecture auto-tuning method (Akiba et al., 2019), which incorporates efficient search and pruning strategies, to determine the optimized hyperparameters following advanced frameworks (Snoek et al., 2012; Koch et al., 2018; Akiba et al., 2019). As the search algorithm, we use covariance matrix adaptation evolutionary strategies (CMA-ES) (Hansen et al., 2003). The search objective in our experiment is set to $\mathcal{M}_\text{P}$, and the parameters that are searched over include top-k, nucleus-η, typical-ϕ, temperature T, and repetition penalty r.

Table 8. Experimental results of precision, recall, and Hamming distance under different ranking methods. All results are reported on a single trial.

Method	$\mathcal{M}_{\mathrm{P}}(\%)(\uparrow)$	$\mathcal{M}_{\mathrm{R}}(\%)(\uparrow)$	$\mathcal{M}_{\mathrm{H}}(\downarrow)$
Baseline	37.0	76.5	19.614
Perplexity $\div$ Zlib	37.1	76.0	20.191
Perplexity $\times$ Zlib	37.1	76.3	20.368
Cumprod	36.6	39.8	20.127
High confidence	40.6	77.3	19.518
Surprised patterns	37.1	75.5	20.072

Table 9. Experimental results of precision, recall, and Hamming distance for compatibility analysis. All results are reported on a single trial.

	$\mathcal{M}_{\mathrm{P}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{R}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{H}}\downarrow$
Baseline(GPT-Neo 1.3B)	19.5	65.6	26.948
Context win + High confidence(GPT-Neo 1.3B)	46.8	77.5	17.144
Baseline(GPT-Neo 2.7B)	33.5	76.6	20.921
Context win + High confidence(GPT-Neo 2.7B)	54.8	81.5	13.621

Table 6 contains the detailed search parameter settings. We also provide the baseline parameters as initial values to the search algorithm to speed up convergence. The number of search rounds is limited to 1,000. The experimental results are shown in Table 5. Simply using the best parameters outlined in Section. 5.1 with k=5, η=0.6, ϕ=0.6, T=0.4, r=1 yields a precision of 48.8%. Implementing auto-tuning results in a 37% improvement over baseline, and auto-tuning performs slightly better than the hypermeter manual section.

6. Improved Suffix Ranking

Following the generation of multiple suffixes, a ranking process is carried out in which less likely suffixes are eliminated using perplexity $\mathcal{P}$ as a metric. However, our statistical analysis in Figure 5 reveals that the ground-truth sentences do not consistently have the lowest perplexity values. Thus, we propose additional ranking metrics to address this disparity.

6.1. Sentence-Level Criteria

Zlib. The entropy of a text, as determined by the Zlib compression algorithm (Gailly & Adler, 2004) using the number of bits, is a quantitative indicator of the sequence's information content. We use the ratio of the perplexity of a given sentence as computed by the GPT-Neo model, and the Zlib entropy of the same sentence as a metric for membership inference, as outlined in the work of Carlini et al. (2021). Besides, we investigate the potential utility of producting perplexity and Zlib entropy, as both metrics tend to decrease when the model shows a high degree of confidence in its predictions. Both metrics produce only a marginal improvement in the overall performance of the membership inference task, as shown in Table 8.

Table 10. Experimental results of precision, recall, and Hamming distance for compatibility analysis. All results are reported on a single trial.

	$\mathcal{M}_{\mathrm{P}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{R}}(\%)\uparrow$	$\mathcal{M}_{\mathrm{H}}\downarrow$
Context win + Beams=2	46.7	77.5	17.154
Auto-tuning + Beams=2	46.4	76.2	17.331
Context win + Auto-tuning	46.5	77.5	17.370
Context win + High confidence	46.8	77.5	17.144

Cumprod. We consider alternative metrics such as the cumprod, which depicts the tandem probability of a generation sentence as $\mathcal{L}_c = (\prod_{n=0}^{N} \log p(x_n|x_0, .., x_{n-1}))^{-N}$. However, the effect of tandem probability on precision is marginal and has a negative effect on $\mathcal{M}_{\mathrm{R}}$.

6.2. Token-Level Criteria

Reward on high confidence. The presence of a high degree of confidence in memorized data is one of the defining features of the phenomenon known as the'memorization effect' (Goodfellow et al., 2016; Zhang et al., 2021; Arpit et al., 2017). We propose implementing a strategy that rewards suffixes with high-confidence tokens based on this insight. If the sentence contains confident tokens, the possibility of the generated token is higher than a threshold, and the difference between the generated token and other token is higher than a threshold, we rank it higher. Specifically, for the token x_n in a generated suffix, if the probability of is higher than a threshold 0.9, then we subtract a given number 0.1 from the score of suffix s_i (the original score for s_i is its perplexity). This trick makes a noticeable improvement as seen in Table 8.

Encouraging surprised patterns. According to recent studies (Holtzman et al., 2019), human text generation frequently exhibits a pattern in which tokens with high perplexity are intermittently included, as opposed to consistently selecting tokens with low perplexity. To address this problem, we propose a simple solution that encourages the presence of surprised tokens (high perplexity tokens) by calculating the perplexity of a generated prompt based on only the majority tokens:

$$\mathcal{L}_s = \frac{1}{|\hat{\mathcal{X}}|} \sum_{x_n \in \hat{\mathcal{X}}} \log p(x_n|x_{[0:n-1]}), \qquad (12)$$
$$\hat{\mathcal{X}} = \{x_j | \mu - 3\sigma < \log p(x_n|x_{[0:n-1]})) < \mu + 3\sigma\},$$

where μ and σ denotes the mean and standard of the $p(x_n|x_{[0:n-1]})$ in a batch. The inclusion of surprised tokens in a generation does not have a negative impact on overall sentence perplexity when using this method, thereby increasing the likelihood of their selection during membership inference. As demonstrated in Table 8, the improvement is relatively marginal.

6.3. Compatibility Analysis

It has been discovered that accumulating the aforementioned tricks does not always result in a proportionate increase in performance. We investigate the interactive effects of combining various techniques in a targeted and efficient manner, in order to establish a viable baseline method. The results are shown in Table 10. The application of multiple of the aforementioned techniques yields no significant improvement in performance. Even when using a beam width of 2 and maintaining the same settings as in the previous experiments, the results do not show a significant improvement. The empirical findings presented in this section suggest that to achieve optimal results, a more deliberate and strategic combination of various methods is required.

We also evaluate the methods on GPT-Neo 2.7B in Table 9 and draw the conclusion that a larger pre-trained model yield better results.

7. Conclusion and Future Work

We investigate a dozen techniques for extracting training data from LMs. These techniques make minor changes to the generation strategies and ranking strategies. Our empirical findings show that several of these tricks improve significantly, while interactions between different tricks are more subtle than expected. Besides, the empirical results show that commonly used versatile methods for general text generation are not always effective for extraction tasks. In future works, explortion in developing compatible techniques in data extraction is preferred. Techniques in both suffix generation and ranking suffix may be combined and explored in an end-to-end fashion. In addition, further investigation to gain a deeper understanding of the underlying mechanisms that contribute to the superior performance of certain techniques in this paper are encouraged.

References

Akiba, Takuya, Sano, Shotaro, Yanase, Toshihiko, Ohta, Takeru, and Koyama, Masanori. Optuna: A next-generation hyperparameter optimization framework. In *ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

Arpit, Devansh, Jastrzkebski, Stanislaw, Ballas, Nicolas, Krueger, David, Bengio, Emmanuel, Kanwal, Maxinder S., Maharaj, Tegan, Fischer, Asja, Courville, Aaron, Bengio, Yoshua, and Lacoste-Julien, Simon. A closer look at memorization in deep networks. In *International Conference on Machine Learning (ICML)*. PMLR, 2017.

Black, Sid, Gao, Leo, Wang, Phil, Leahy, Connor, and Biderman, Stella. Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow, 2021.

Brown, Tom, Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared D, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Carlini, Nicholas, Liu, Chang, Erlingsson, Úlfar, Kos, Jernej, and Song, Dawn. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2019.

Carlini, Nicholas, Tramer, Florian, Wallace, Eric, Jagielski, Matthew, Herbert-Voss, Ariel, Lee, Katherine, Roberts, Adam, Brown, Tom, Song, Dawn, Erlingsson, Ulfar, et al. Extracting training data from large language models. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2021.

Carlini, Nicholas, Ippolito, Daphne, Jagielski, Matthew, Lee, Katherine, Tramer, Florian, and Zhang, Chiyuan. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022.

Choquette-Choo, Christopher A, Tramer, Florian, Carlini, Nicholas, and Papernot, Nicolas. Label-only membership inference attacks. In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

Fan, Angela, Lewis, Mike, and Dauphin, Yann. Hierarchical neural story generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Feldman, Vitaly. Does learning require memorization? a short tale about a long tail. In *Proceedings of the ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.

Fredrikson, Matt, Jha, Somesh, and Ristenpart, Thomas. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the ACM SIGSAC conference on computer and communications security (ACM CCS)*, 2015.

Freitag, Markus and Al-Onaizan, Yaser. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pp. 56–60, Vancouver, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3207. URL https://aclanthology.org/W17-3207.

Gailly, Jean-loup and Adler, Mark. Zlib compression library, 2004.

Ganju, Karan, Wang, Qi, Yang, Wei, Gunter, Carl A, and Borisov, Nikita. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the ACM SIGSAC conference on computer and communications security (ACM CCS)*, 2018.

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

Gong, Neil Zhenqiang and Liu, Bin. You are who you know and how you behave: attribute inference attacks via users' social friends and behaviors. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2016.

Gong, Xueluan, Wang, Qian, Chen, Yanjiao, Yang, Wang, and Jiang, Xinchang. Model extraction attacks and defenses on cloud-based machine learning models. *IEEE Communications Magazine*, 58(12), 2020.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep learning*. MIT press, 2016.

Hansen, Nikolaus, Müller, Sibylle D, and Koumoutsakos, Petros. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary computation*, 11(1), 2003.

Hayes, J, Melis, L, Danezis, G, and De Cristofaro, E. Logan: membership inference attacks against generative models. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*. De Gruyter, 2019.

He, Yang, Rahimian, Shadi, Schiele, Bernt, and Fritz, Mario. Segmentations-leak: Membership inference attacks and defenses in semantic image segmentation. In *European Conference on Computer Vision (ECCV)*. Springer, 2020.

Hilprecht, Benjamin, Härterich, Martin, and Bernau, Daniel. Monte carlo and reconstruction membership inference attacks against generative models. *Proc. Priv. Enhancing Technol.*, 2019(4), 2019.

Hinton, Geoffrey E., Vinyals, Oriol, and Dean, Jeffrey. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL `http://arxiv.org/abs/1503.02531`.

Holtzman, Ari, Buys, Jan, Forbes, Maxwell, Bosselut, Antoine, Golub, David, and Choi, Yejin. Learning to write with cooperative discriminators. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Holtzman, Ari, Buys, Jan, Du, Li, Forbes, Maxwell, and Choi, Yejin. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2019.

Hu, Hongsheng, Salcic, Zoran, Sun, Lichao, Dobbie, Gillian, Yu, Philip S, and Zhang, Xuyun. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54(11s), 2022.

Jagielski, Matthew, Thakkar, Om, Tramer, Florian, Ippolito, Daphne, Lee, Katherine, Carlini, Nicholas, Wallace, Eric, Song, Shuang, Thakurta, Abhradeep, Papernot, Nicolas, et al. Measuring forgetting of memorized training examples. *arXiv preprint arXiv:2207.00099*, 2022.

Jayaraman, Bargav, Ghosh, Esha, Inan, Huseyin, Chase, Melissa, Roy, Sambuddha, and Dai, Wei. Active data pattern extraction attacks on generative language models. *arXiv preprint arXiv:2207.10802*, 2022.

Juuti, Mika, Szyller, Sebastian, Marchal, Samuel, and Asokan, N. Prada: protecting against dnn model stealing attacks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019.

Kandpal, Nikhil, Wallace, Eric, and Raffel, Colin. Deduplicating training data mitigates privacy risks in language models. *arXiv preprint arXiv:2202.06539*, 2022.

Kaya, Yigitcan and Dumitras, Tudor. When does data augmentation help with membership inference attacks? In *International Conference on Machine Learning (ICML)*. PMLR, 2021.

Kenton, Jacob Devlin Ming-Wei Chang and Toutanova, Lee Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Keskar, Nitish Shirish, McCann, Bryan, Varshney, Lav R, Xiong, Caiming, and Socher, Richard. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Koch, Patrick, Golovidov, Oleg, Gardner, Steven, Wujek, Brett, Griffin, Joshua, and Xu, Yan. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *ACM International Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

Lehman, Eric, Jain, Sarthak, Pichotta, Karl, Goldberg, Yoav, and Wallace, Byron C. Does bert pretrained on clinical notes reveal sensitive data? In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2021.

Leino, Klas and Fredrikson, Matt. Stolen memories: leveraging model memorization for calibrated white-box membership inference. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2020.

Liu, Jinghui, Capurro, Daniel, Nguyen, Anthony, and Verspoor, Karin. "note bloat" impacts deep learning-based nlp models for clinical prediction tasks. *Journal of biomedical informatics (JBI)*, 133, 2022.

Meister, Clara, Pimentel, Tiago, Wiher, Gian, and Cotterell, Ryan. Locally typical sampling. *Transactions of the Association for Computational Linguistics (TACL)*, 2022.

Naseri, Mohammad, Hayes, Jamie, and De Cristofaro, Emiliano. Toward robustness and privacy in federated learning: Experimenting with local and central differential privacy. *arXiv e-prints*, 2020.

Parisot, M, Spagnuelo, D, et al. Property inference attacks on convolutional neural networks: Influence and implications of target model's complexity. *Proceedings of the International Conference on Security and Cryptography*, 2021.

Radford, Alec, Wu, Jeffrey, Child, Rewon, Luan, David, Amodei, Dario, Sutskever, Ilya, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8), 2019.

Rezaei, Shahbaz and Liu, Xin. On the difficulty of membership inference attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Sablayrolles, Alexandre, Douze, Matthijs, Schmid, Cordelia, Ollivier, Yann, and Jégou, Hervé. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning (ICML)*. PMLR, 2019.

Salem, Ahmed, Zhang, Yang, Humbert, Mathias, Berrang, Pascal, Fritz, Mario, and Backes, Michael. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

Shokri, Reza, Stronati, Marco, Song, Congzheng, and Shmatikov, Vitaly. Membership inference attacks against machine learning models. In *IEEE symposium on security and privacy (SP)*. IEEE, 2017.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, 2012.

Song, Liwei and Mittal, Prateek. Systematic evaluation of privacy risks of machine learning models. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2021.

Thomas, Aleena, Adelani, David Ifeoluwa, Davody, Ali, Mogadala, Aditya, and Klakow, Dietrich. Investigating the impact of pre-trained word embeddings on memorization in neural networks. In *International Conference on Text, Speech, and Dialogue (TSD)*. Springer, 2020.

Tople, Shruti, Sharma, Amit, and Nori, Aditya. Alleviating privacy attacks via causal learning. In *International Conference on Machine Learning (ICML)*. PMLR, 2020.

Tramèr, Florian, Zhang, Fan, Juels, Ari, Reiter, Michael K, and Ristenpart, Thomas. Stealing machine learning models via prediction {APIs}. In *Proceedings of the USENIX Conference on Security Symposium (SEC)*, 2016.

Wang, Yijue, Wang, Chenghong, Wang, Zigeng, Zhou, Shanglin, Liu, Hang, Bi, Jinbo, Ding, Caiwen, and Rajasekaran, Sanguthevar. Against membership inference attack: Pruning is all you need. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2021.

Welleck, Sean, Kulikov, Ilia, Roller, Stephen, Dinan, Emily, Cho, Kyunghyun, and Weston, Jason. Neural text generation with unlikelihood training. In *International Conference on Learning Representations (ICLR)*, 2019.

Wolf, Thomas, Debut, Lysandre, Sanh, Victor, Chaumond, Julien, Delangue, Clement, Moi, Anthony, Cistac, Pierric, Rault, Tim, Louf, Rémi, Funtowicz, Morgan, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, 2020.

Wu, Bang, Yang, Xiangwen, Pan, Shirui, and Yuan, Xingliang. Model extraction attacks on graph neural networks: Taxonomy and realisation. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security (ACM ASIA-CCS)*, 2022.

Yeom, Samuel, Giacomelli, Irene, Fredrikson, Matt, and Jha, Somesh. Privacy risk in machine learning: Analyzing the connection to overfitting. In *IEEE computer security foundations symposium (CSF)*. IEEE, 2018.

Zhang, Chiyuan, Bengio, Samy, Hardt, Moritz, Recht, Benjamin, and Vinyals, Oriol. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3), 2021.

A. Appendix

A.1. Guidelines

Recently, there has been a growing emphasis on the importance of extracting training data for language models. Through our examination of various techniques in this paper, we have discovered that certain tricks do not yield as favorable results as compared to the baseline approach, or may entail substantial computational expense. Therefore, we propose the following guidelines for the development of data extraction methods for language models.

1. Usability: In order for a data extraction method to be considered practical, it is expected have a low threshold for adoption and be easy to use. For example, not requiring additional datasets, having a straightforward implementation without the need for complex modules, and being easily adaptable for various applications.

2. Effectiveness: The data extraction methods are expected to demonstrate a good performance on the given datasets.

3. Efficiency: It is essential that the methods are computationally efficient, utilizing limited computation cost. Additionally, it is desirable for the methods to incorporate small yet effective modules. Extra LMs improve the performances, but we expect the method to replace them with small modules without a noticeable performance trade-off.

4. Perniciousness. As an adversarial task to data protection, we expect data extraction methods from language models to be pernicious since a pernicious strong extraction attack can better evaluate the risk of information leakage accurately, equivalent to a higher lowerbound for potential leakage.

We would like to emphasize that the guidelines aim to enhance the success rate of the training data extraction methods and thus provide a more accurate evaluation of the privacy information leakage of LMs, instead of posing pernicious attack methods.

A.2. Typical Failure Cases of Training Data Extraction

In this section, we observe the classical instances of failures that occur in extracting training data and classify them into two categories, as outlined in Table 12. The first category of failure pertains to instances where the generated suffix is incorrect from the tokens in the middle portion, resulting in subsequent tokens being generated inaccurately. The second category of failure pertains to instances where only a limited number of tokens in the suffix do not correspond with the ground truth and the latter tokens are correctly generated.

The method of look-ahead is motivated by the second category of failure encountered during training data extraction. In addition, it is evident from the examples of both successful and failure cases of data extraction that, both the prefix and the suffix are not complete sentences. For instance, the prefix can begin with a punctuation mark '.', or a broken sentence like 'WARRANTY OF ANY KIND'. It is due to the prevalent techniques of truncating and padding training language materials. In light of this phenomenon, this paper presents the methods of dynamic context window and dynamic position shifting.

In contrast to failure cases, Table 11 presents a selection of successful examples of extraction. These examples may serve to provide a tangible understanding of the task of training data extraction.

A.3. Histogram without Interploration

A.3.1. Sentence perplexity histogram

To investigate the distribution of sentence perplexity of generated suffixes and ground truth suffixes, we use histograms to visualize their discrepancies. We compute the sentence perplexity of both ground truth and multiple generated suffixes, and then calculate the rank of ground truth perplexity and draw a histogram based on this in Figure 5, where the x-axis is calculated as

$$\mathcal{S}(\mathcal{P}_{gt}, \{\mathcal{P}_{s_i}\}), \tag{13}$$

where $\mathcal{S}(a, b)$ returns the rank of a in a set b, $\mathcal{P}_{gt}$ is the perplexity of the ground truth suffix of a given prefix p, s_i denotes the i-th generated suffix of a given prefix p.

A.3.2. Token logits histogram

To investigate the distribution of logits values between generated tokens and ground truth tokens, we use histograms to visualize their discrepancies. A histogram of token logits for both ground truth suffixes and generated suffixes is presented in Figure 6, wherein a clear discrepancy in the logits distribution can be observed. Furthermore, in order to gain insight into the distribution of logits values across different positions, histograms of logits values for various positions are also provided in Figure 7, 8, and 9.

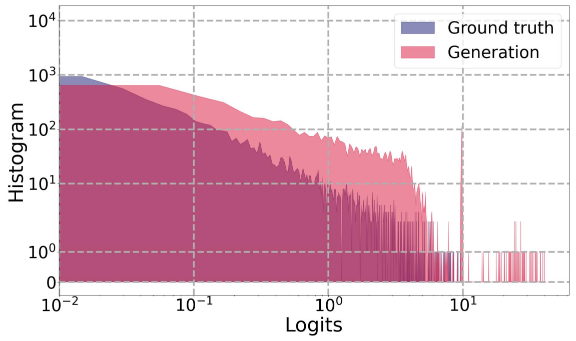

Figure 6. Histogram of token logits. The histogram depicts the distribution of logit values obtained from 1,000 suffixes, each containing 50 tokens.

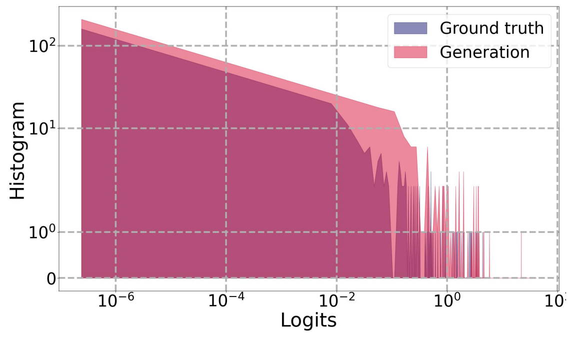

Figure 7. Logits values of 0-th (the first) generated token. The histogram depicts the distribution of logit values obtained from the 0-th token of 1,000 suffixes.

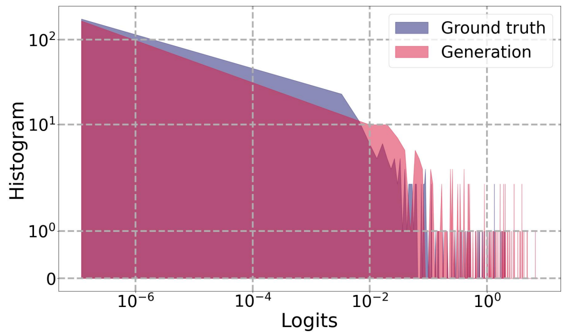

Figure 8. Logits values of 24-th (the middle) generated token. The histogram depicts the distribution of logit values obtained from the 24-th token of 1,000 suffixes.

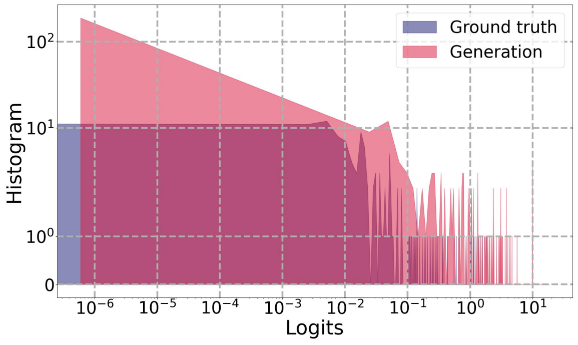

Figure 9. Logits values of 49-th (the last) generated token. The histogram depicts the distribution of logit values obtained from the 49-th token of 1,000 suffixes.

A.4. Takeaways

we conclude with some takeaways as follows,

1. In the two stages (suffix generation and suffix ranking) of training data extraction, suffix generation influences the extraction precision more than suffix ranking.

2. Dynamic context window is a significantly more useful trick than others.

3. Sampling strategies are highly sensitive to hyperparameters.

4. We recommend using fewer than or equal to two beams, as the number of beams significantly increases memory requirements but brings degressive precision improvements.

5. The look-ahead mechanisms augments the precision of the presently generated token, and the degree of enhancement is contingent upon the number of future tokens taken into account.

Table 11. Training data extraction successful cases. The generated suffix equals the ground truth suffix in both length and tokens.

Prefix	/* * Copyright 2003-2006 Sun Microsystems, Inc. All Rights Reserved. * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER. * * This code is free software;
Generated Suffix	MERCHANTABILITY or * FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License * version 2 for more details (a copy is included in the LICENSE file that * accompanied this code).
Prefix	Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the &quot;Software&quot;), to deal in the Software without restriction, including without limitation the rights to
Generated Suffix	INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR

A.5. Negative Societal Impacts

In this study, various techniques for extracting training data are investigated. It is imperative to note that these methods may be utilized for nefarious purposes. The focus of this paper is on a specific type of training data extraction, namely the target training data extraction, which requires a prefix in the training dataset. If an attacker has access to the training data of a language model or the ability to replicate similar sentences to the training data, it is possible for them to recover the sentences of the training data, as demonstrated in Table 11.

We mitigate the negative ethics impacts in this work, where we focus on a specific LM GPT-Neo, which uses Pile as its training dataset. Note that Pile is already a public dataset obtained from mainly academic or professional sources instead of private sources.

The primary intended users of training data extraction include institutions and organizations that aim to investigate the privacy and security characteristics of language models. However, it is crucial to acknowledge that these methods may also be used by malicious attackers. In the short term, some LMs present a risk of leaking information. And in the long term, we appeal to the language community to mitigate the risk and develop secure language models which exhibit resistance to attacks including membership inference attacks, training data extraction attacks, etc.

Table 12. Training data extraction examples. Red tokens indicate the mismatch tokens between the generated suffix and ground truth suffix.

Prefix	.
	//
	// Copyright (c) 2008-2011 Texas Instruments Incorporated. All rights reserved.
	// Software License Agreement
	//
	// Texas Instruments (TI) is supplying this software for use solely and
	// exclusively on TI

Generated Suffix	NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
	// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
	// Redistribution and use insource and binary forms, with

Ground Truth Suffix	NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
	// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
	// CIRCUMSTANCES, BE LIA

Prefix	. No comments: Friends Dis-complainer The Great Change is published whenever the spirit moves me. Writings on this site are purely the opinion of Albert Bates and are subject to a Creative Commons Attribution Non-
Generated Suffix	(NC): You may not use this work for commercial purposes. Commercial use is not permitted without the express written consent of the copyright holder.
Ground Truth Suffix	(NC): You may not use this work for commercial purposes. Share Alike (SA): If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one. Nothing in
Prefix	# Permission is hereby granted, free of charge, to any person # obtaining a copy of this software and associated documentation # files (the ”Software”), to deal in the Software without # restriction,
Generated Suffix	WARRANTY OF ANY KIND, # EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF # MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND # NONINFRINGEMENT
Ground Truth Suffix	WARRANTY OF ANY KIND, # EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES # OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND # NONINFRINGEMENT
Prefix	//// //// THIS SOFTWARE IS PROVIDED “AS IS” AND WITHOUT ANY //// //// EXPRESS OR IMPLIED WARRANTIES, INCLUDING
Generated Suffix	DAMAGES //// //// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS //// //// OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUS
Ground Truth Suffix	DAMAGES //// //// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE //// //// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;