LLMs Leak Training Data Beyond Verbatim Memorization: Extraction via Membership Decoding

Zitai Chen
National University of Singapore
zitai@comp.nus.edu.sg

Reza Shokri
National University of Singapore
reza@comp.nus.edu.sg

Abstract

Extracting training data from large language models (LLMs) is a serious privacy breach that exposes (potentially private) data without data owners' consent. Existing extractions follow the generation-then-audit paradigm, where the greedy decoding method in generation limits the extraction scope and only verbatim memorized data is under audits. A majority of partially memorized member data (around 90%) remains unexplored, of which LLMs could memorize almost all tokens but fail to rank the training token to top-1 at some positions. To measure the degree of such a partial memorization, we introduce a new notion of memorization, k/n-correction, by the number of non-top-1 tokens k in a suffix of length n. This notion quantifies the memorization in a fine-grained manner. Experiments show that models memorize more with smaller average k values as the model size increases.

To extract the partial memorization, we propose a new decoding method, named Membership Decoding, by introducing membership information in the generation process. The Membership Decoding method is a plug-and-play replacement for standard decoding that requires only black-box token probabilities. We formalize the data extraction problem as a next member token prediction problem. Accordingly, we propose a new token-level membership inference method by leveraging likelihood from reference models, shifting the generation from the original token distribution to the member token distribution. Extensive experiments show that partial memorization is much more prevalent than verbatim memorization, and membership decoding can extract previously unextractible partially-memorized sequences, succeeding in correcting sequences with up to $k = 2$ non-member tokens in the suffix of length $n = 10$. The proposed attack demonstrates the potential privacy risks in partial memorization.

Keywords

Large Language Models, Training Data Extraction, Membership Inference, Privacy, Memorization

1 Introduction

Training data extraction attacks are able to reconstruct training data from a target model. Such an exposure of training data without the data owner's consent is a severe privacy breach. The privacy breach is escalated when the extracted content possesses actual privacy-sensitive information like personally identifiable information (PII).

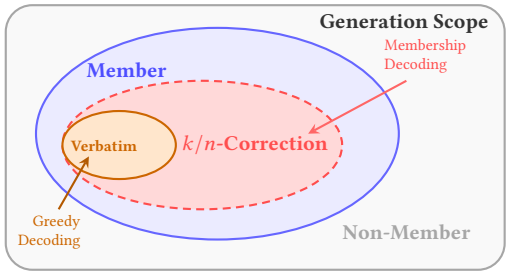

Figure 1: Relationship between the scope of generation, member data, k/n-correction, and verbatim memorization. Greedy decoding extracts verbatim memorization, while MIA can identify all member data. To fill this gap, we propose Membership Decoding to extract member data on partial memorization $k > 0$.

In the context of large language models (LLMs), this privacy risk is further enhanced because more data increases the chance of PII being present, and larger models memorize more. More training data have enabled LLMs to have remarkable capabilities in text generation [3], reasoning [12], agent workflow [7], but also in memorization [5]. Thus, it is crucial to understand the memorization behavior of LLMs to estimate the potential extraction risks. Recent works [3, 6, 31] have quantified the memorization by verbatim memorization, where LLMs complete a training prefix with the same training suffix with greedy decoding. To extract verbatim memorized sequences, attackers follow the generation-then-audit pipeline: generate a suffix by prompting a training prefix first, and then audit its membership in the training dataset by performing membership inference attacks (MIAs) [5, 6].

However, the risk estimation based on verbatim memorization is not comprehensive. Partial memorization could be extracted with advanced decoding strategies. The extraction risk is not a binary state where verbatim memorization is unsafe and otherwise is safe, but a spectrum where the suffixes are partially memorized to different degrees with varying levels of exposure. The partial memorization indicates that LLMs could memorize almost every training token in suffix but fail to recall at some positions during greedy decoding. The correct suffix token may rank at the Top-5 of the generation distribution with high confidence, yet remains "hidden" because a more generic, frequent, or safe token narrowly edges it out for the Top-1 position. For example, Alice could make mistakes at the first try of recalling her 9-digits identity number. These partially memorized data that is not captured by greedy decoding, as depicted in Figure 1. This requires a new privacy notion to define and measure the partial memorization.

Proceedings on Privacy Enhancing Technologies 2026(4), 651–665

https://doi.org/10.56553/popets-2026-0139

index	k/n-correction	Prefix → **Generation**	k-extractible
		Alice's ID is 123 456 789	$k = \lvert$ Alice's ID is 123 456 7$\rvert = 11$ tokens
p_1	$k = 2, n = 9$	Alice's ID is → **000 000 000**	$\lvert$Alice's ID is $\rvert = 4 < k \Rightarrow$ Safe
p_2	$k = 1, n = 8$	Alice's ID is 1 → **23 456 666**	$\lvert$Alice's ID is 1 $\rvert = 5 < k \Rightarrow$ Safe
p_3	$k = 1, n = 3$	Alice's ID is 123 456 → **666**	$\lvert$Alice's ID is 123 456 $\rvert = 10 < k \Rightarrow$ Safe
p_4	$k = 0, n = 2$	Alice's ID is 123 456 7 → **89**	$\lvert$Alice's ID is 123 456 7 $\rvert = 11 = k \Rightarrow$ Risky

Table 1: Examples for memorization notions under greedy decoding. Red denotes generating non-member tokens and Green denotes the corrected member token. k/n-correction can capture the risk accurately for different given prefixes, while k-extractible does not differentiate.

Furthermore, the generation-then-audit pipeline is not optimal for extraction. The generation step is the bottleneck of extraction that partial memorization is not generated. In existing extraction studies, greedy decoding limits the extraction scope to verbatim memorized sequences. To address the limitation in greedy decoding, advanced decoding strategies introduce bias and randomness to the decoding process, such as beam search decoding [27], biasing the decoding with the later token probability, and top-K sampling [9], randomizing the decoding within top-K tokens. They could expose partially memorized sequences to extraction attacks. For example, to recall the correct identity number, Alice can use auxiliary messages like state code or try multiple times. Recent works [13, 31] have studied the memorization amplification brought by randomness as multi-shot extraction. However, multiple generation trials are computational expensive for LLMs. In this paper, we explore the direction of bias, introducing membership information to bias the generation.

In this work, we explore the key research question as follows:

> *Can LLMs generate partially memorized training data with a membership-guided decoding strategy?*

To answer this question, we first quantify the partial memorization with a new notion, **k/n-correction**, measuring how many tokens of a suffix are not clearly memorized by LLMs given its training prefix. Specifically, given a suffix of length n, we compute k as the number of targeted suffix tokens that are not the most probable in the next token prediction distribution. The generation of these k member tokens must take additional information to correct the distribution for member token generation. Verbatim memorization is a special case of k/n-correction where $k = 0$ and can be extracted without additional membership guidance. A stronger extraction attack should achieve higher accuracy in extracting seriously partial memorization, i.e., more k/n-correction sequences with larger $k > 0$. A clear relationship among generations is shown in Figure 1.

After defining the goal of extraction, we propose a novel extraction pipeline to address the challenge of introducing membership information to bias generation. The key idea is to reframe the training data extraction problem as a training data completion problem in an autoregressive way, which naturally align with the normal usage of LLMs. At each token generation step, we shift the generation goal from generating the most probable token to generating the most probable member token. Accordingly, we bias the token distribution towards the member token distribution with help of membership inference attacks. This generation process guided by MIAs at every step can keep the generation staying on the path of member tokens, even when the greedy path diverges. The extraction pipeline is shifted from "generation-then-audit" to "audit-in-between-generation". It also shifts the focus from post-hoc analysis to real-time integration of membership during generation.

Following this motivation, we propose the **Membership Decoding** framework from greedy decoding by altering the token selection criterion with MIAs. This framework allows us to leverage MIA scores to calibrate the prediction distribution at each generation step. Accordingly, to address the special need in differentiating member from non-member tokens, we propose a novel token-level membership inference attack based on maximizing the posterior probability of observing the member prefix. In this MIA score, we hypothesize that the member token has the largest "probability advantage" if the model has been trained on the token. The probability advantage is compared to well-generalized reference models. As such, we compute the posterior distribution of the token rather than the original likelihood distribution, which is more discriminative for membership inference. Crucially, the membership information can be inferred from standard, black-box accessible with token likelihoods.

In this work, the proposed membership decoding framework unifies the extraction attacks and allows for extraction with more advanced MIAs. Our contributions are threefold:

(1) We provide a fine-grained partial memorization metric, k/n-correction. With this metric, we find that partial memorization dominates 90% of training sequences, and structured code/math data is more vulnerable to extraction.
(2) We introduce a novel extraction pipeline, Membership Decoding, that shifts the paradigm from "generation-then-audit" to "audit-in-between-generation". This proof-of-concept demonstrates the feasibility of expanding extraction studies to capture partial memorization.
(3) We propose a novel token-level membership attack to differentiate member tokens from non-member tokens during Membership Decoding. Experiments demonstrate the effectiveness of our approach in extracting some $k = 1$ and $k = 2$ sequences that greedy decoding misses entirely.

2 Related Work

2.1 Membership Inference Attack

Membership inference attacks (MIAs) aim to decide whether a specific data point was included in the training dataset of a target model [4, 23, 29, 30, 32]. MIAs are first introduced for auditing machine learning algorithms [23]. This method and subsequent works

are based on the training of many reference models to calibrate the MIA score for the target model behavior.

MIAs on LLMs have been considered a challenging problem [8]. Methods can be categorized into training-based MIA and training-free MIA. Training-based MIAs, like LiRA [4, 21], train many IN models and OUT models to calibrate the MIA scores for a specific target sample, which is expensive for LLM. Training-free MIAs [16, 18, 22, 28] analyze LLM signals themselves. *Min-k%* [22] and its variant *Min-k%++* [33] calibrate the k-lowest token likelihood, which is insensitive to token changes out of the minimum k. *DC-PDD* [35] calibrates token probability with token frequency distribution. Rather than identifying true members, they define non-member detectors by catching outlier signals. *Neighbor-comparison* [16] catches the overfitting signal by referring to neighbor data, which is inefficient, involving massive generations from the reference model. *RECALL* [28] computes the likelihood shift amount with a non-member prefix.

All existing MIA methods act as evaluation-only tools, catching the average member token signal for sequence MIAs. They fail to provide an active and discriminative signal on the member token to guide generations. Thus, they are not suitable for generative guidance. In this work, we design a token-level membership score to distinguish member tokens from non-member tokens to lead the training data generation.

2.2 Training data extraction

Data extraction attacks aim to recreate a training data point from a target model. It reveals severe privacy risks of leaking sensitive and personally identifiable information (PII). Unintentional memorization is recognized as a main source of extraction. Existing extraction attacks can be categorized as training-based and training-free attacks.

Training-free attacks perform membership inference attacks after massive generations. [6] performed loss-based MIA on text sequences generated with greedy decoding. The extractible sequences are limited to verbatim memorized sequences. [13, 24] expanded the generation scope by probabilistic sampling, taking the top-k sampling method to explore members. [31] adjusted the probability distribution of tokens with repetition penalty and temperature to allow diversity in massive generation. However, performing membership inference attacks on massive generations is expensive and inefficient for extraction.

Training-based attacks train assistant models to extract training data from target models [20, 25]. [20] proposed to learn a model adapter in the form of a soft prompt to extract training data. [25] proposed to learn a soft prompt generator to dynamically enhance the extraction capability. However, these training-based attacks require white-box access to the target model for gradient computation. They also need a large amount of training data to train the adapter or generator. They are not applicable in black-box settings where only output logits or probability distributions are available.

Apart from the extraction of exact suffixes, approximate extractions relax the constraints to allow partial matching [2], n-gram matching [14], and approximate string matching [15]. In this work, we focus on the exact suffix extraction, which is more challenging and more useful in real-world applications.

3 Threat Model and Problem Definition

3.1 Memorization

We first define membership in LLMs, where we consider all possible prefixes of a full training sequence as members due to the next token prediction training objective. Then we define the k/n-correction notion to capture the degree of partial memorization, which provides more fine-grained information than the existing k-extractible notion. The k/n-correction can estimate the extraction risk of a sequence and create risk profile of different models for comparison.

Definition 3.1 (Membership in LLMs)**.** A sequence x is a member of a model f if x is the prefix of a sequence in the training dataset of f. Otherwise, it is a non-member.

Definition 3.2 (k-extractible)**.** A suffix s is k-extractible if it is generated by the model f when prompted with a prefix p of length k, and $[p, s]$ is a member.

k-extractible is the first memorization notion in LLMs [6], which measures the number of tokens needed in the prefix to extract the sequence by greedy decoding. The greedy decoding strategy chooses the token with the highest likelihood as a continuation, which is the default setting for **one-shot extraction** in most prior works [5, 6]. Sampling introduces randomness for multiple suffix generations, known as **multi-shot extraction**. A member sequence is (n, p)-extractible [13] if it is generated in n trials with probability p, measuring memorization under probabilistic decoding. However, these privacy notions fail to answer the question of how much of a member sequence is actually memorized by the model. To address this, we introduce the k/n-correction notion, which counts the number of tokens in the suffix that are not decoded by greedy decoding on the given prefix. It captures the degree of memorization by k, and a non-trivial suffix length n, as a side note, indicating the usefulness of the extracted sequence.

Definition 3.3 (k/n-correction)**.** Given a specific member sequence $x = [p, s] \in D$ where p is the prefix and the suffix of length n as

$$s = [s_1, \ldots, s_n],$$

we define the generation path of *greedy decoding* as

$$\hat{s} = [\hat{s}_1, \ldots, \hat{s}_n],$$

where

$$\hat{s}_t = \arg\max_{v \in V} Pr_f(v|p, s_{<t}).$$

We claim that the prefix p is k/n-correction (k tokens correction out of n tokens suffix) if

$$k = \sum_{t=1}^{n} \mathbb{I}(\hat{s}_t \neq s_t).$$

The k/n-correction provide more fine-grained memorization analysis than k-extractible, which provides a binary view on the extraction risk by only locating the last incorrect token. Our notion considers the total number of incorrect tokens in the suffix, which is more informative for understanding the memorization behavior of LLMs. The k-extractible is sensitive to the location of the last incorrect token, while k/n-correction considers the total number of incorrect tokens in the suffix of a given prefix. We provide a

concrete example in Table 1 to explain the difference. Given the member sequence *"Alice's ID is 123 456 789"*, k-extractible consider prefix p_3 *"Alice's ID is 123 456 "* with suffix *"666"* from greedy decoding as a safe prefix. However, such a safety is fragile where adversaries can extract the full suffix by correcting only one token "6" to "7" as p_4. Such a partial memorization sequence can be extracted easily, as observed in [13]. While k/n-correction can reflect such a vulnerability by flagging $k = 1$. More importantly, a shorter prefix doesn't necessarily indicate a smaller extraction risk: the shorter prefix p_2 shares a similar risk as p_3.

Risk profile of LLMs. The k/n-correction can further estimate the extraction risk of a dataset by averaging the k values of the sequence, allowing comparison between models on the same training dataset. In experiments, as shown in Figure 4, we find that k decreases on average as model size increases, showing heavier memorization in larger models in the Pythia model family [3]. This can serve as the risk profile of LLMs. Additionally, the extraction success rate decreases as k grows. It suggests that the k/n-correction notion is valid and more practical in estimating real-world extraction risk and memorization.

Prefix length is not the adversary's choice. The extraction adversary acquires prefix from a partially leaked or a PII-masked document. Shortening the prefix with fewer tokens doesn't provide any advantage in extraction, and could even increase the difficulty of extracting a suffix with a larger k/n-correction value. Thus, we consider the prefix as a given condition rather than a variable in the definition. The adversary's goal is to extract the suffix on the given prefix, which is more practical in real-world applications.

3.2 Problem Definition

We define the data extraction attack in language models as a training data completion problem. The goal is to design a mechanism to generate the target suffix on the given prefix:

Definition 3.4 (Training Data Extraction)**.** Given a target language model f trained for next token prediction on dataset D and a prefix p from a member sequence $x = [p, s] \in D$, the goal is to design a mechanism g to generate the target suffix s:

$$g(p, f) = s. \tag{1}$$

The extraction mechanism g can be deterministic or probabilistic, like greedy decoding [6]

$$g := \arg\max_i (f(p)_i),$$

and sampling [13, 31]

$$g := Sampling_i(f(p)_i).$$

However, none of the existing attacks takes membership information into the generation. In this paper, we explore the member information of the prefix to calibrate the next token prediction distribution during the sequence generation process.

We evaluate the extraction of the suffix from the prefix as follows:

Definition 3.5 (Success Extraction)**.** An extraction attack g is successful if the generated suffix $g(p, f) = \hat{s}$ is identical to the target suffix s, i.e., $\prod_i^n \mathbb{I}(s_i = \hat{s}_i) = 1$.

Targeted data extraction. We focus on targeted data extraction, where the adversary seeks a specific s known to follow p, i.e., $x = [p, s] \in D$. The targeted data extraction is more privacy-related, where adversaries often have some prior knowledge about the target data (e.g., a leaked prefix of "Alice's ID is 123") and want to extract the remaining information (e.g. a complete ID number "456 789"). It also allows us to evaluate the extraction risk of specific sequences, which is more informative for understanding the memorization behavior of LLMs. Untargeted data extraction [6] always starts from a short prefix that is followed by multiple suffixes, which is less risky and less informative. But we emphasize that any extraction of training data violates data owner's privacy even when the content is not confidential. In experiments, we take prefix with more than 100 tokens to lower the possibility of multiple member suffixes, and suffix with $n = 10$ token to ensure that the extracted sequence is substantial. However, extracting partially memorized suffixes at any length is valuable for auditing; it can flag anomalies in default generation by identifying common phrases, or uncover privacy risks by exposing unique, sensitive information.

3.3 Threat Model

Following the common practice, we consider the threat model defined in the literature [6].

Victim Definition. The victim provides black-box access to the target language model and returns the logprobs of a list of tokens on the query sequence. In a real-world scenario, the victim can be a commercial LLM provider, which provides API access to the model. We further assume that the victim discloses only the top-H tokens for privacy protection, as the commercial practice with $H = 20$. The target model is trained with the next token prediction objective.

Although a top-H token return policy offers robust defense by omitting member tokens from the attack search space, it is not infallible. Our findings reveal that the member token frequently appears within the top-5 or top-10 rankings, enabling the successful extraction of partially memorized sequences.

Adversary's Capabilities. Adversary can query any sequence $x_{<t}$ and receive the logprobs of the top-H tokens. The weight and intermediate prediction of LLM are hidden. An adversary can sample the member prefix $p = x_{<t}$ with an unknown suffix s to initiate the extraction. The adversary can also have access to a reference model h with the same tokenizer as the target model.

As a proof of concept, we demonstrate that our black-box attack remains highly practical despite requiring a reference model for calibration. This setup does not depend on knowing the target model's exact training distribution. For open models, our experiments utilize a downscaled model from the same family. For closed-source commercial APIs, we propose a viable workaround: an attacker can construct a reference model by pairing public datasets with the provider's official tokenizer, which is publicly accessible for billing calculations. This underscores that the availability of a reference model is a highly realistic threat assumption. Our empirical tests on open weights show that membership signals are robustly structural: even a miniature 70M model can effectively calibrate outputs from a 12B target model. This finding exposes a critical security

Algorithm 1 Greedy Decoding

Require: Prefix $x_{<m}$, target suffix length n, model f, vocabulary V
Ensure: Generated member suffix $x_m, x_{m+1}, \ldots, x_{m+n-1}$
for $t = m$ **to** $m + n - 1$ **do**
  Construct candidate set:
    $\{c_i^t = [x_{<t}, v_i]\}_{i=1}^{|V|}$
  Compute likelihood score for each candidate c_i^t:
    $Pr(c_i^t, f)$
  Select the token that maximizes the score:
    $x_t \leftarrow \arg\max_{v_i} Pr(c_i^t, f)$
end for

return $x_m, x_{m+1}, \ldots, x_{m+n-1}$

Algorithm 2 Membership Decoding

Require: Prefix $x_{<m}$, target suffix length n, model f, vocabulary V
Ensure: Generated member suffix $x_m, x_{m+1}, \ldots, x_{m+n-1}$
for $t = m$ **to** $m + n - 1$ **do**
  Construct candidate set:
    $\{c_i^t = [x_{<t}, v_i]\}_{i=1}^{|V|}$
  Shortlist with Top-H tokens:
    $\{c_{h_i}^t\}_{i=1}^{H}$ where $h_i = \arg\max_{j \in \{1,\ldots,H\}} Pr_f(c_{h_j}^t, f)$
  Compute MIA score for each candidate c_i^t:
    $Score_{MIA}(c_i^t, f)$
  Select the token that maximizes the score:
    $x_t \leftarrow \arg\max_{v_i} Score_{MIA}(c_i^t, f)$
end for

return $x_m, x_{m+1}, \ldots, x_{m+n-1}$

gap, suggesting that commercial LLMs could face similar extraction risks using minimal public resources.

Adversary Objective. The goal of the adversary is to extract the member suffix given the member prefix. A stronger attack can achieve higher extraction accuracy on k/n-correction sequences with larger $k > 0$.

Evaluation. The evaluation is performed on individual k values, which is more informative to understand the memorization behavior of LLMs. A larger k indicates how poorly a suffix is memorized by the model with respect to the next token prediction distribution. A stronger attack should be able to extract those sequences even when the model poorly memorizes. **Our goal is to extract member data that is k/n-correction with $k > 0$, expanding generation scope to partial memorization.**

In this work, we focus on the one-shot extraction setting [6], where the adversary can only generate one candidate suffix for accuracy measurement. But the multi-shot extraction with sampling [13] is considered as baseline in experiments.

4 Method

In this section, we explain our membership decoding formulation to address the training data extraction problem. The graphical illustration of membership decoding is presented in Figure 2. We also propose a new score for context-aware token-level membership inference, distinguishing a member token from a set of non-member tokens in the vocabulary.

Challenges. The first challenge is that the candidate space of possible member suffixes is huge. It grows exponentially in the suffix length n: $|V|^n$. The second challenge is that the membership inference attack score is not accurate when the data is not verbatim memorized. The score is computed as the likelihood of suffix given prefix $P_f(x_{m,\cdots,x_{m+n-1}}|x_{<m}) = \prod_{t=m}^{m+n-1} P_f(x_t|x_{<t})$.

4.1 Membership Decoding

Greedy Decoding. Before expressing our decoding process, we briefly introduce the default generation process of greedy decoding. First, we create a set of candidate continuations $c_i^t = [x_{<t}, v_i]$ from the vocabulary V for the current prefix $x_{<t}$. Then, we compute the next token likelihood probability for each candidate c_i^t: $Pr_f(v_i|x_{<t})$. Finally, the token with the highest likelihood is selected from the candidate set c_i^t as the next token in greedy decoding. The generation process continues until the maximum length is reached. The overall process is shown in Algorithm 1. The greedy decoding process is a local optimization that selects the most likely token at each step, aiming for a global optimization of generating the most likely sequence. This process decomposes the most probable global (sequence-level) generation problem into local (token-level) generation subproblems. It reduces the search space to the vocabulary size at each step. However, it is not optimal for training data extraction attacks. The member token may not be the most probable (Top-1) at each step, especially when the data is partially memorized. No membership information of the prefix is utilized to guide the generation.

Insights. Inspired by the greedy decoding, we address the first challenge by formulating the training sequence extraction problem as a training data completion problem. We decompose a sequence-level extraction problem into a sequential token-level membership inference attack. The key insight is that the prefix of any length of a member sequence is also a member. We can perform MIAs on the next member token prediction given the member prefix. In particular, due to the auto-regressive nature of LLMs, the training objective of a training sequence is a series of token predictions with cross-entropy loss l:

$$L(x_{<m+n}) = \sum_{t=1}^{m+n-1} l(x_t|f(x_{<t})).$$

Each prefix $x_{<t}$ serves as a member of the target sequence $x_{<m+n}$ in the training dataset. The model is trained to predict the next member token x_t given its prefix $x_{<t}$. Thus, we can perform token-level membership inference attacks during generation to complete the member suffix.

Membership Decoding. Compared to greedy decoding, we alter the usual "most probable" token generation to "member" token generation for the training data extraction attacks as presented in Algorithm 2. At each generation step t, a member sequence is

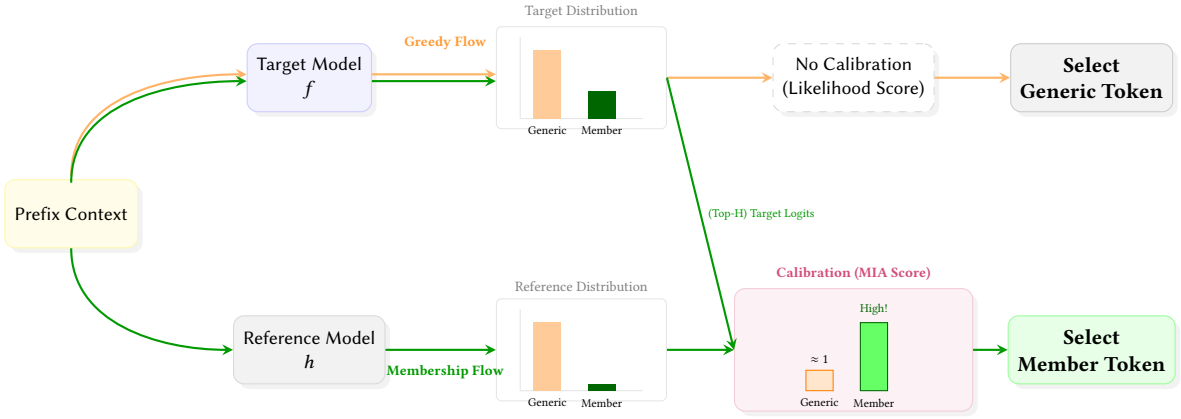

Figure 2: The membership decoding process and calibration step, compared with greedy decoding.

identified from the candidate set by membership inference attacks. First, we construct the member candidate set by appending a token from the vocabulary V to the member prefix $x_{<t}$ as Step 2. This step can reduce the search space to the size of the vocabulary $|V|$ at each step and grows linearly with the suffix length n, as the greedy decoding does. Second, we compute the MIA score for each candidate c_i^t as Step 6. The MIA score is designed to infer the membership information of the candidate sequence. Finally, we select the token with the highest MIA score as the next member token in Step 8. The generation continues until the member sequence is completed.

We define the extraction attack mechanism g as a next member token prediction function that takes the prefix $x_{<t}$ and the model f to predict the next member token x_t until the member sequence $x_{<m+n}$ is completed.

$$g(x_{<t}, f) = x_t, \quad \forall t = m, m+1, \cdots, m+n-1.$$

In each next token prediction, the mechanism g leverages the membership score to select a member token as the next token. This process calibrates the default token distribution to favor member tokens, bringing minor overhead to the default decoding process. Thus, we name this framework Membership Decoding, aiming to pop up the member token during the generation. This framework allows us to explore different membership attacks by varying the membership score in Step 6. In summary, we propose the Membership Decoding framework to address the training data extraction problem. This is a local (token-level) optimization that aims for a global (sequence-level) extraction.

To adhere to real-world API constraints on privacy protection (such as the top-20 logprobs in OpenAI's chat completion API), we apply the constraint of disclosing the Top-H tokens to shortlist our candidate set c_i^t as Step 4. This API constraint is proposed to alleviate the information leakage from the model by limiting the number of disclosed logprobs. On one hand, a clear protection is set up when the member token is excluded from the shortlisted candidate set, significantly reducing the adversary's search space from full vocabulary size $|V|$ to a manageable subset of size 20. It lowers the ceiling of success for all extraction attacks relying on logprobs. On the other hand, by only "correcting" tokens within this high-probability window, we ensure that the extraction yields meaningful tokens while mitigating the risk of nonsensical tokens due to the false positives from MIAs. This approach ensures that we are auditing the model's memorization signals rather than exploiting random noise in the low-probability tail.

In the next section, we explain our design for the membership score. Greedy decoding is a special case of membership decoding. It implements Loss-based MIA [30], computing membership score as the likelihood of a candidate token given the prefix:

$$Score_{Loss}(f, c^t) = Pr_f(x_t | x_{<t}).$$

However, the Loss-based MIA score fails to calibrate the hardness of the target sample [4]. Without calibration, the membership scores are not accurate, especially when the data is partially memorized.

4.2 Maximize a Posterior as MIA Score

In this part, we propose our MIA score computation method in Membership Decoding. We first explain the motivation of our score design from the perspective of maximizing a posterior probability. Then we detail the computation of the posterior probability with Bayes' rule. Finally, we explain the approximation of the likelihood and evidence terms in Bayes' rule with the next token prediction probability and reference models, respectively.

Insights. Given $x_{<t}, \forall t < n$ are members of the model f, we aim to evaluate the probability of observing $(x_{<t}, x_t = v_i)$ as a member sequence. To address the second challenge, we need to calibrate the membership score to better distinguish the member sequence from the non-member sequences. The key insight is that if $\hat{v}$ is the member continuation of $x_{<t}$, the posterior probability of observing the member prefix $x_{<t}$ should be higher than given a non-member continuation v_i.

Posterior. We define MIA score as the probability of "f is trained on $x_{<t}$" given "f is trained on $(x_{<t}, x_t = v_i)$":

$$Score_{MIA}(f, x_{<t}, x_t = v_i) = P(f, x_{<t} | x_t = v_i).$$

However, we cannot directly compute this probability of a candidate sequence c_i^t given the prefix $x_{<t}$. It requires computing the probability of observing "f is trained on $x_{<t}$" over all possible training datasets, which is intractable. To address this challenge, we apply

Bayes' rule to compute the posterior probability of a candidate sequence c_i^t given the prefix $x_{<t}$:

$$v_m = \arg\max_{v_i} P(f, x_{<t} | x_t = v_i), \tag{2}$$

$$\text{(with Bayes)} \quad = \arg\max_{v_i} \frac{P(f, x_{<t}) P(x_t = v_i | f, x_{<t})}{P(x_t = v_i)}, \tag{3}$$

$$\text{(Independent)} \quad \propto \arg\max_{v_i} \frac{P(x_t = v_i | f, x_{<t})}{P(x_t = v_i)}, \tag{4}$$

where the prior $P(f, x_{<t})$, the probability of "f is trained on $x_{<t}$", is independent to candidates and left out.

Likelihood. The likelihood is the probability of observing the next token v_i given the member prefix $x_{<t}$ and model f. We approximate likelihood with next token prediction probability of prefix $x_{<t}$ on target model f:

$$P(x_t = v_i | f, x_{<t}) = Pr_f(x_t = v_i | x_{<t}). \tag{5}$$

Evidence. The evidence $P(x_t = v_i)$ is the total probability of "observing the token v_i as the next token given the prefix $x_{<t}$". It is computed by marginalizing all possible models h evaluated on the prefix $x_{<t}$:

$$P(x_t = v_i) = \sum_{h, x_{<t}} P(h, x_{<t}) P(x_t = v_i | h, x_{<t}). \tag{6}$$

To approximate this evidence computation, we resort to open-weight LLMs as the reference models. However, it is challenging to find reference LLMs h that are trained on the same target sequence $x_{<m+n}$, i.e., IN models. Otherwise, the estimation is biased to the models that are not trained on sequence $x_{<m+n}$, i.e., OUT models. We calibrate this bias by assuming that the estimation on IN models is interpolation between overfitting models and OUT models as RMIA [32] and overfitting models return probability 1 on all next token predictions:

$$P(x_t = v_i | h_{IN}, x_{<t}) = \alpha \cdot P(x_t = v_i | h_{OUT}, x_{<t}) + (1 - \alpha) \cdot 1 \in [0, 1], \tag{7}$$

where α is a hyper-parameter to control the calibration strength. The likelihood of IN models is overestimated as overfitting models when $\alpha = 0$, and underestimated as OUT model when $\alpha = 1$. The overfitting assumption of probability 1 is an idealization case. This assumption represents the expected "overfitting gap" between member and non-member. Finally, the evidence is approximated with one reference model as follows:

$$P(x_t = v_i) \approx \frac{1}{2}((1 + \alpha) \cdot P(x_t = v_i | h, x_{<t}) + (1 - \alpha)). \tag{8}$$

Hyper-parameter Selection. We take $\alpha = 0.5$ throughout the experiment to avoid the bias between underestimation biasing to OUT models and overestimation biasing to overfitting models. This value is found empirically in the experiment, and a detailed analysis of the effect of α is in the experiment section. $\alpha = 0.5$ assumes the IN model assigns at least 50% probability to the candidate token regardless of the reference OUT model's prediction. Ablation study on α is presented in the experiment section, which shows that the performance is not sensitive to α when $\alpha \in [0.25, 0.75]$.

Overall, we compute our token-level membership score as follows:

$$Score_{MIA}(c_i^t, f) = \frac{2 \cdot Pr_f(x_t = v_i | x_{<t})}{(1 + \alpha) \cdot Pr_h(x_t = v_i | x_{<t}) + (1 - \alpha)} \tag{9}$$

In summary, we define the membership score by calibrating the next token prediction probability with the evidence of observing token v_i as the member continuation over all possible models h. Greedy decoding [6] takes no calibration as a Loss-based MIA method. Reference-based [18] methods calibrate with the likelihood of the reference model, which is $\alpha = 1$ in our case. Similarly, the token frequency method calibrates with the approximate total probability by the token frequency distribution from the reference dataset. Our membership decoding framework unifies these methods by varying the membership score, allowing us to evaluate different membership inference attacks in finding the member token.

5 Experiment

In this section, we first evaluate the memorization issues in LLM via the k/n-correction notion. Then we evaluate the effectiveness of membership decoding in generating partially memorized data.

5.1 Experiment Setup

Datasets. We evaluate on the MIMIR benchmark [8], which is created from the Pile dataset [10] for MIA evaluation. We take 7 datasets from different domain: "Pile CC", "ArXiv", "PubMed Central", "HackerNews", "DM Mathematics", "GitHub", "Wikipedia", each of which contains 1,000 member sequences. And we present the average result across all datasets as "Average". For each member sequence, we define the suffix as the last $n = 10$ tokens, and all the leading tokens as the prefix. All the prefixes have more than 100 tokens and are less likely to overlap with the other sequences with a different suffix. The suffix is unique in the extraction experiment. The dataset contains no privacy-sensitive data and serves as a testbed for extraction attacks. We emphasize that the exposure of training data without the owner's consent is a severe privacy breach, even if the content is not sensitive.

Models. For all the experiments, we take the transformer models introduced in Pythia [3]. The target models are Pythia-1b, 1.4b, 2.8b, 6.9b, 12b, and the reference model is Pythia-70m, consistent with the literature [22, 28] as a weakly-fit baseline of a non-member evidence estimator. Smaller models tend to memorize less of the training data [2] and can be considered as a well-generalized OUT model. For Pythia-12b, the prefix length is limited to 300 tokens due to the memory limitation. We also present the result of the OPT family [34] and OLMo family [11]. All experiments are conducted on two NVIDIA RTX-TITAN GPUs with 24GB of memory. The model weights and datasets are loaded from HuggingFace [26]. No randomness is introduced in both generation and membership inference attacks. All results are reproducible.

Baseline. We compare four decoding baselines:

- **Loss** [30] (Greedy decoding): It selects the token with the highest probability $Pr_f(v_i | x_{<t})$, defining MIA score as the likelihood as Eqn. 5.

- **Freq**: It calibrates the likelihood with the token frequency in the reference dataset $Pr_f(v_i|x_{<t})/Pr_D(v_i|x_{<t})$, where frequency is computed from reference dataset [35].
- **Diff**: It selects token with the highest likelihood difference $Pr_f(v_i|x_{<t}) - Pr_h(v_i|x_{<t})$, catching the likelihood gains compared to the reference model.
- **Ratio** [18]: It selects token with the highest likelihood ratio $Pr_f(v_i|x_{<t})/Pr_h(v_i|x_{<t})$, i.e., $\alpha = 1$ in Eqn. 9, catching the likelihood increase rate compared to the reference model.
- **Our**: We select the token that maximizes the posterior as Eqn. 9.

The other token-insensitive MIA methods like Min-k% [22] are not applicable in our membership decoding framework, as they fail to distinguish member tokens from non-member tokens at every step.

Metric. The performance is measured by the sequence-level exact match defined in Definition 3.5. We evaluate the performance in the subgroup of k values according to the k/n-correction. The sequences in group $k = 1$ have 1 incorrect token during greedy decoding. The performance is presented as the percentage of sequences with an exact match in each group, as accuracy.

Settings. To isolate the evaluation of the next member token prediction task for MIAs, we construct two settings: **Hard** and **Easy**. The **Hard** setting requires performing the next $n = 10$ member tokens extraction. While the **Easy** setting requires performing 1 member tokens prediction at the failure case of greedy decoding. Evaluation data is constructed by the failure prefix of greedy decoding, where the token with the highest probability is not a member token.

5.2 Memorization with k/n-correction

In this section, we answer the following research questions regarding the memorization level of training data in LLMs and its relationship with model scale and data domain:

- **Q1:** Does the proposed notion of memorization k/n-correction reflect the memorization level of training data in LLMs?
- **Q2:** How does the memorization level of training data change across different model scales and data domains?

Partial memorization is more prevalent than verbatim memorization. We evaluate the k value for each sequence by querying the target model with the prefix $[p, s_{<t}], t = 1, \cdots n$ as Definition 3.3. The k is the number of incorrect token predictions in the greedy decoding process. Computing this notion of memorization is as efficient as the generation process. To evaluate the extraction risk across data domains, we present the k-value distribution of Pythia-6.9b in Figure 3 and specify the average percentage in the legend. The detailed percentage of sequences with each k value is shown in Table 6 in Appendix. On average, only 10.5% of sequences are verbatim memorized ($k = 0$), and 89.5% of the training dataset are partially memorized. The high risk partial memorization sequence with $k = 1$ and $k = 2$ accounts for a significant portion (18.7%) of the dataset. This result demonstrates the importance of studying the partial memorization problem in this work.

The k/n-correction notion reflects the memorization level of training data in each domain. The higher percentage of sequences with small k values indicates a higher memorization level. The "GitHub" dataset presents a significantly higher memorization issue, where more than 40% sequences are verbatim memorized. While the "HackerNews" dataset is mainly composed of $k = 4, 5, 6$ sequences, suggesting a lower memorization level. The k/n-correction notion indicates that the Pythia-6.9b model memorizes more "GitHub" samples than "HackerNews" samples, which is supported by their respective MIA vulnerability presented in the paper [8]. Such results from the cross-validation suggest that the k/n-correction indeed captures the memorization level of the training data and is able to reflect the vulnerability to membership inference attacks.

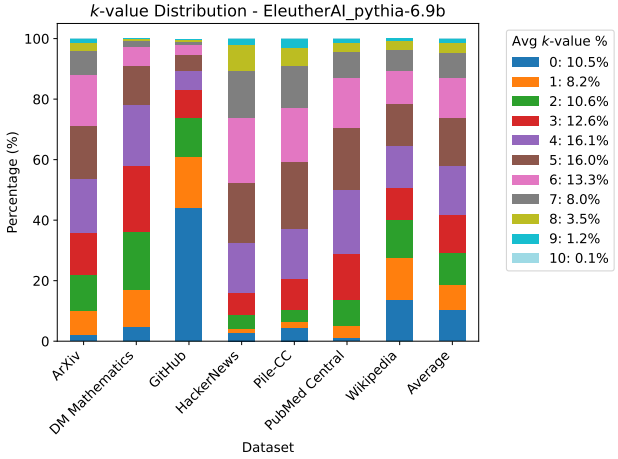

Figure 3: k-value distribution on 7 datasets on Pythia-6.9b

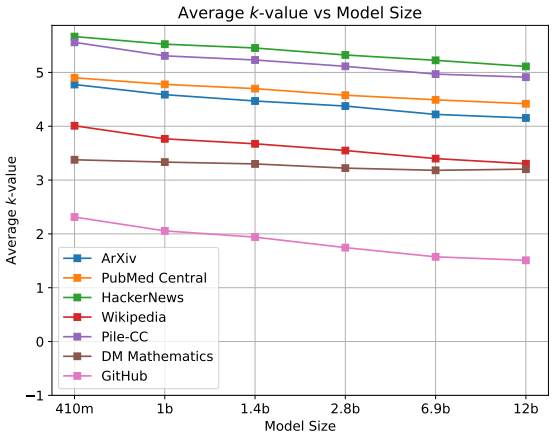

Figure 4: Average k-value across multiple model scales

Models at a larger scale memorize more training data. k/n-correction notion can serve as a memorization profile to compare the memorization level of different models. We compute the average k value of the datasets at different model scale in Figure 4. We introduce a smaller model with 'Pythia-410m' for a full-scale comparison. The

average k value of a dataset is computed by $\sum_k k * Percentage@k$. The average k value presents a clearly decreasing trend when the model scales up. This phenomenon suggests that larger models memorize more training data than smaller models, which has also been observed in previous works [5]. It supports us to take smaller models as reference models in membership decoding, as they memorize less training data.

Structured data is more vulnerable to partial memorization. The average k value of each data domain preserves the same order (HackerNews > PubMed > Mathematics > GitHub) across model scales. It suggests that the memorization level of the data domain is consistent for all model sizes, and the cause is inherent in the data domain. The GitHub and Mathematics domains have a lower average k value than other domains, as shown in Figure 3 and 4. The average k value of GitHub and Mathematics datasets is around 3, while the average k value of other datasets is around 5. It suggests that structured data (code and math) might be more susceptible to the partial memorization problem than natural language.

Privacy Implications. From the above comparison across data domain and model scales, the k/n-correction notion provides a detailed memorization profile. It can be used to audit the model and identify the vulnerable data domain and model scale. This information is helpful for practitioners to understand the memorization and guide the model training and deployment.

5.3 Easy Setting (Single Token)

In this section, we isolate the effectiveness of the scoring function itself without the compounding error of a long generation. It evaluates the case where the attacker has prior knowledge of the candidate token set but greedy decoding returns an outlier, like those in constrained decoding [17]. We answer the following research question related to the member token prediction power of membership inference attacks:

- **Q3:** How well can the membership inference attacks extract the member token when not ranking the first?

Dataset construction. The test dataset is constructed by the failure cases of greedy decoding in 7 datasets, where member token does not rank the first in the next token prediction. All the test sequences are 1/1-correction sequences with member prefixes following the attack assumption that the prefix is a member.

Metric. We measure the membership inference attack performance in terms of the Top@H accuracy ($H = 1, 2, 4, 8, 16, 20$) as Definition 3.5 with $n = 1$. The Top@H accuracy measures whether the member token appears within the top H highest-probability tokens. It reflects the extraction power of the MIAs even if the best member guess is wrong. For every sequence in this setting, greedy decoding fails for Top@1 by definition. We acknowledge that testing examples in Easy setting are biased, as the most probable token is a non-member. However, none of the testing methods leverages this bias explicitly as prior knowledge to improve the performance. Thus, the comparison across methods is fair and valid.

Hidden member tokens in the next token prediction distribution are still extractible, even with a lower probability. All four membership scores have non-zero accuracy on Top@1 (around 5% on "ArXiv") as shown in Figure 5. The membership inference attacks can still identify the member token from the target model logprobs, even when it is hidden in the probability distribution with a lower probability. The consistent success across all data domains shows that MIAs are robust to domain changes and the extraction risk is universal. These results prove that the membership guidance in decoding is able to keep the generation path on the member token path.

Our MIA score can rescue greedy decoding on Top@1 while maintaining a comparable accuracy on Top@H as greedy decoding. It indicates that our MIA score is on the right track of improving the order of member token. While Ratio and Freq fail to improve the ranking consistently at Top@2 and Top@4. This improvement also demonstrates the effectiveness of the total probability calibration in Eq. 8.

The extraction of k/n-correction sequences is challenging. On one hand, our results provide a proof-of-concept that the extraction of member tokens is possible. It shows the vulnerability of partially memorized data to training data extraction attacks. But the low accuracy ($< 10\%$) in single token correction experiments shows the difficulty of this problem for the current methods. On the other hand, the accuracy of Top@20 is around 80%, indicating that the full suffix extraction in Hard setting is not always possible as the member token is not presented in the candidate set of membership decoding. Our results illustrate that the real-world API constraint on limiting the returned tokens does provide certain protection against the training data extraction attacks.

5.4 Hard Setting (Full Suffix)

In this section, we evaluate the membership decoding on generating the next $n = 10$ consecutive member tokens on each group k of k/n-correction sequences. We answer the following research question:

- **Q4:** Can LLMs generate partially memorized training data with the membership guidance?

The extraction difficulty increases with a larger k, as more non-member tokens score a lower loss and more chance of the member token being out of the top 20 tokens. With a larger k, membership decoding is required to correct more tokens to keep the generation on the member path. The Loss baseline is left out as the group k is defined by it.

Partially memorized sequence is vulnerable to training data extraction attacks. As shown in Table 2, our method successfully extracts sequences of k/n-correction with $k = 1, 2$. Even though the numbers seem low, they are impossible for greedy decoding to achieve. This result illustrates a qualitative breach of privacy that greedy decoding entirely misses in one-shot extraction. Partially memorized data can still be extracted with the membership guidance, as the member token is still extractible from the target model logprobs. The shorter member prefix from a k-extractible sequence is not safe from training data extraction attacks. Our result provides a proof-of-concept demonstration that the privacy risk of the training data is not only from verbatim memorization, but also from partial memorization.

The extraction of partially memorized data is more challenging than verbatim memorized data. The results from the Pythia show

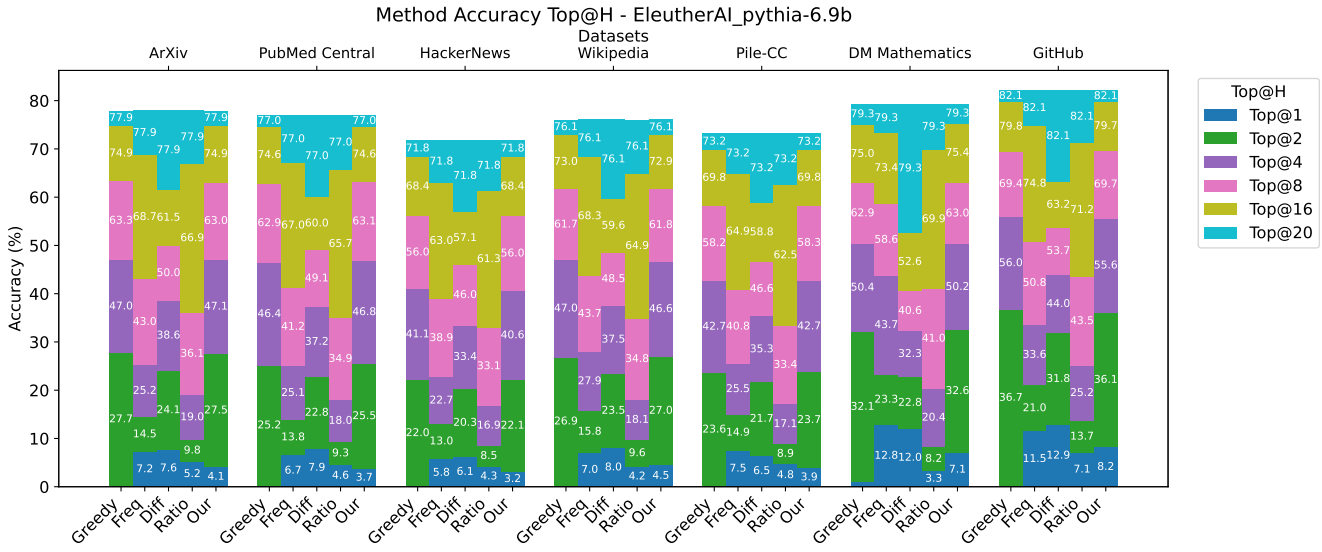

Figure 5: Easy: Result of single token MIA on each k/n-correction sequence on Pythia-6.9b. The membership information can recover the member token when the greedy decoding fails.

that the extraction success rate decreases in log-scale as k increases. The extraction rate on GitHub is around 10% for $k = 1$ sequences and 1% for $k = 2$ sequences. This result reflects the nature of partial memorization. The drop-off performance on $k > 3$ is because the larger the value a sequence has, the less is memorized by the model, and the weaker the membership signal is. The membership signal could be too noisy to guide the generation for a large k value, which leads to a low extraction success rate. This is also verified in the later defense experiment in Figure 8, where the recoverability of the larger k sequences decreases significantly. The result also suggests that the membership guidance in decoding is more effective for sequences with smaller k values.

The total probability calibration improves the overall utility of the membership score. Compare to the reference-model-based method Ratio, our MIA score takes the total probability calibration in Eq. 8, which improves the extraction accuracy from 0 to 8.2% on $k = 1$ sequences on GitHub Pythia-6.9b, and from 0 to 1.3% on $k = 2$ sequences. This shows that the total probability calibration is effective in improving the overall utility of the membership score. The assumption in the Ratio method that IN models share the same likelihood as the OUT model underestimates the likelihood of IN models and decreases the accuracy of the membership score.

The performance drop on $k = 0$ sequences is because the failure of MIAs during the generation process. Since the k value and the location of the non-member token are unknown to the attacks, MIA is performed at every step to determine the member token. For $k = 0$ sequences, the member token is always the most probable token, and the MIA score is expected to be the highest. However, the MIA score can fail to rank the member token as the top one at some step, which leads to a non-member token being generated and the generation path going off the member path.

5.5 OPT and OLMo Results

To further evaluate the effectiveness of membership decoding under different model architectures and datasets, we also conduct experiments on OPT [34] and OLMo [11] families. The reference model is the smallest in each family, i.e., OPT-125m and OLMo-1.3b. For the OPT family, we take three model scales: OPT-1.3b, OPT-2.7b, and OPT-6.7b. We evaluate on four datasets: HackerNews, Pile-CC, PubMed Central, and ArXiv in Pile [10]. For the OLMo family, we take OLMo-7b as the target model and evaluate on a Dolma dataset example subset [11]. We sample 5000 examples from the Dolma_v1.6 example dataset and limit the prefix length to 90 tokens due to the memory limitation. The other experiment settings are consistent with those in the previous experiment.

The experiment results are shown in Table 3 for OPT and Table 4 for OLMo. Among all three membership scores, our score outperforms others on both model families. It suggests that the total probability calibration in Eq. 8 is effective across different model architectures and datasets. The results also indicate that membership decoding is a general framework for training data extraction attacks. Also, the privacy risk from partially memorized data is prevalent across different model architectures and datasets.

5.6 Multi-shot Extraction

In this section, we compare our method with multi-shot extraction methods: sampling decoding and beam search decoding. For TopK sampling, we sample 10 samples and take perplexity as their MIAs scores. For beam search, we set the beam size to 4 and return the one with the highest probability. The other experiment settings are consistent with those in the Hard setting.

We present the accuracy averaged across datasets in Figure 6 with k-value ranging from 0 to 2. On the one hand, the multi-shot extraction method does not show significant improvement

Table 2: Hard setting: Extraction accuracy (%) on Pythia

k	Method	Wikipedia					Pile-CC					ArXiv					PubMed Central				
		1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b
0	Freq	6.9	12.0	9.2	11.8	12.2	18.8	24.2	30.3	18.2	34.7	0.0	0.0	0.0	0.0	0.0	30.8	27.3	33.3	30.8	26.7
	Diff	11.9	9.3	12.5	11.8	16.2	25.0	33.3	36.4	45.5	57.1	6.7	10.5	23.8	17.4	31.8	0.0	0.0	0.0	15.4	6.7
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Our	86.1	84.3	80.8	83.8	79.7	93.8	90.9	90.9	86.4	93.9	73.3	68.4	71.4	78.3	72.7	76.9	81.8	83.3	92.3	93.3
1	Freq	1.5	0.8	0.7	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	0.0	0.0	0.0	0.0	0.0	0.0
	Diff	0.0	0.0	0.7	2.9	3.6	0.0	0.0	9.1	10.0	5.6	1.9	5.0	1.5	5.2	4.5	0.0	0.0	0.0	0.0	2.2
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Our	4.4	5.4	8.0	10.0	6.5	0.0	0.0	9.1	20.0	11.1	0.0	5.0	3.0	2.6	6.8	0.0	2.4	2.2	2.5	6.7
2	Freq	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0
	Diff	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.9	0.0	0.0	0.0	1.6	0.0	0.0	0.0
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Our	0.0	0.0	0.0	0.0	0.9	2.8	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

k	Method	HackerNews					DM Mathematics					GitHub					Average				
		1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b
0	Freq	32.0	20.0	41.4	37.0	40.0	2.3	0.0	0.0	0.0	0.0	24.3	27.6	30.9	30.8	32.3	16.4	15.9	20.7	18.4	20.8
	Diff	16.0	24.0	24.1	25.9	23.3	2.3	6.8	5.9	10.4	7.8	25.8	30.3	35.7	40.0	44.5	12.5	16.3	19.8	23.8	26.8
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.4
	Our	100.	100.	100.	100.	100.	65.1	68.2	76.5	75.0	76.5	92.3	93.8	94.5	94.1	96.2	83.9	83.9	85.3	87.1	87.5
1	Freq	9.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.5	1.5	0.1	0.1	0.3	0.2
	Diff	9.1	0.0	0.0	7.7	0.0	0.0	0.0	0.0	0.0	0.8	11.6	9.1	10.5	8.4	8.9	3.2	2.0	3.1	4.9	3.7
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Our	9.1	8.3	0.0	7.7	0.0	7.3	7.7	7.5	9.1	9.2	13.8	17.0	16.3	12.0	10.4	4.9	6.6	6.6	9.1	7.2
2	Freq	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1	0.0
	Diff	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.6	0.9	0.8	1.0	0.1	0.5	0.4	0.1	0.1
	Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Our	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.5	1.1	1.6	0.8	3.5	0.8	1.0	0.8	0.2	0.6	0.2	0.4

Table 3: Extraction Accuracy (%) on OPT by k, Method, Dataset

k	Method	HackerNews			Pile-CC			PubMed Central			ArXiv		
		1.3b	2.7b	6.7b	1.3b	2.7b	6.7b	1.3b	2.7b	6.7b	1.3b	2.7b	6.7b
0	Ratio	0.00	0.00	0.00	3.10	0.00	7.70	0.00	0.00	0.00	0.00	0.00	3.70
	Diff	33.3	21.1	29.4	50.0	45.9	64.1	20.0	29.4	33.3	45.7	22.7	14.8
	Our	93.3	89.5	94.1	93.8	89.2	89.7	60.0	70.6	75.0	80.4	86.4	74.1
1	Ratio	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Diff	0.00	0.00	0.00	0.00	6.70	7.70	0.00	2.30	0.00	6.40	5.60	1.60
	Our	0.00	8.30	0.00	0.00	13.3	11.5	6.20	2.30	4.00	7.40	6.50	11.5

compared to our method in a one-shot setting, at the cost of higher computational overhead. On the other hand, the performance drops as k increases, indicating that extracting less memorized data is challenging in general.

5.7 Parameter Analysis on Calibration

In this section, we analyze the impact of calibration parameter α in Eqn. 7. We vary α with $[0, 0.25, 0.5, 0.75, 1]$ and evaluate the extraction accuracy on Pythia models. $\alpha = 1$ underestimates the probability of IN models as the same as OUT models, while $\alpha = 0$ overestimates the IN model by assigning probability of 1 to the member token. The other values of α represent different calibration strengths between these two extremes. We present the average performance of different α values across datasets and k values in Figure 7 and detailed results in Table 7 in Appendix A.1.

Table 4: Extraction Accuracy (%) on OLMo-7b

k	method	Dolma
0	Ratio	0.00
	Diff	8.29
	Our	89.86
1	Ratio	0.00
	Diff	0.40
	Our	4.90

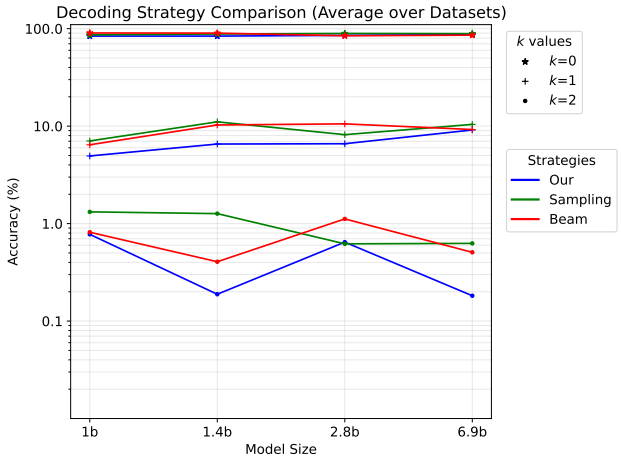

Figure 6: Average performance across datasets on multi-shot extraction.

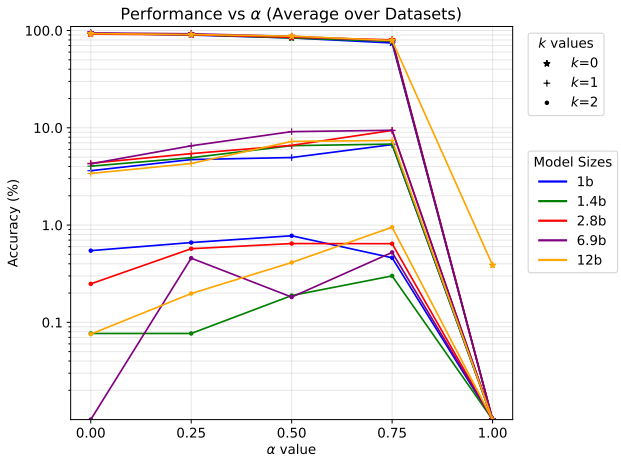

Figure 7: Average performance of different α values across datasets. The performance is not sensitive to α when $\alpha \in [0.25, 0.75]$.

The extraction of verbatim memorization ($k = 0$) and partially memorized data ($(k > 0)$) present a different pattern as alpha increases. The performance shows an increasing trend on $k = 1$ and $k = 2$ sequences and a decreasing trend on $k = 0$ sequences. But the performance drops to 0 when underestimation occurs at $\alpha = 1$. Our initial selection of $\alpha = 0.5$ in avoiding the bias to two extremes allows us to extract partial memorization with a reasonable performance on verbatim memorization. The extraction performance is robust to calibration strength α when $\alpha \in [0.25, 0.75]$.

The principle of selecting α is to achieve a better overall performance on both verbatim memorization ($k = 0$) and partially memorized data ($k > 0$). Given a prefix with unknown k-value, we can formulate the α selection problem as an expected attack success rate (ASR) maximization problem on an auxiliary dataset:

$$\max_{\alpha} \sum_{k=0}^{n} Percentage@k * ASR(k, \alpha) \tag{10}$$

where $Percentage@k$ is the percentage of sequences with k-value in the auxiliary dataset, and $ASR(k, \alpha)$ is the attack success rate on k-correction sequences with calibration parameter α. The optimal α can be computed by grid search on the auxiliary dataset. We provide a detailed computation example in Appendix A.2. Our results can guide the selection of α at different model scales and dataset types in practice.

5.8 Suffix Length Study

Table 5: Suffix length $n = [10, 20, 50]$ on Pythia-2.8b.

Method	k=0			k=1			k=2		
	10	20	50	10	20	50	10	20	50
Freq	20.7	13.1	3.3	0.1	0.0	0.0	0.0	0.0	0.0
Diff	19.8	10.0	6.2	3.1	0.6	0.9	0.4	0.0	0.0
Ratio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Our	85.3	77.4	89.9	6.6	5.4	6.0	0.6	0.3	0.5

In this section, we evaluate the performance of membership decoding on different suffix lengths. We take Pythia-2.8b as the target model, and present the average results in Table 5. The performance of our extraction is robust to the suffix length for $k = 0$ and $k = 1$ sequences. A longer suffix does not increase the extraction difficulty for $k = 1$ sequences, as the model memorizes most of the suffix tokens. The membership signal is strong enough for the attack to identify the memorized member tokens.

6 Discussion

Computational Efficiency. A common concern in iterative decoding strategies is the computational overhead. In Membership Decoding, we calibrate the token distribution at each step, which involves querying both the target and the reference models. While the per-token computational cost is indeed higher than greedy decoding generation, the cost is the same when we consider the cost of the post-hoc MIAs. We spent this budget on the generation process. More importantly, the standard greedy decoding wastes a massive

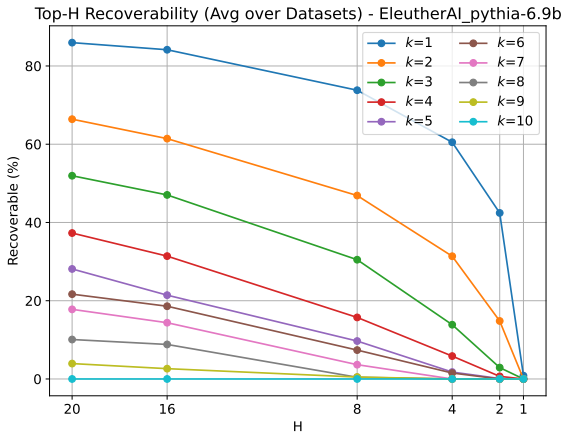

Figure 8: Limit the returned logprobs to H tokens.

amount of computation on generating non-member sequences. In contrast, Membership Decoding's guided search path is much more likely to terminate in a member sequence, making it a more efficient attack in terms of total adversary effort.

Robustness to Reference Model. The choice of reference model can impact the performance of membership decoding. The reference model plays a critical role in estimating the evidence and providing a baseline for the calibration. The reference information provides the necessary inference power to sort out the member token. However, the reference model can be noisy and may not perfectly represent the distribution of non-member tokens. The calibration parameter α can help mitigate the impact of noise by controlling the strength of the calibration. The effectiveness of Pythia-70m as a reference model across different target model scales and data domains proves the robustness of our approach.

6.1 Potential Defense

To mitigate the extraction risk from membership decoding, one must either obscure the necessary MIA signal or increase the required number of corrections (k) such that extraction becomes computationally infeasible. Differential privacy (DP) training [1] can be applied during model training to limit the amount of information about any individual training example that the model can memorize. However, DP often leads to a non-negligible utility trade-off. Model editing techniques [19] can be employed to selectively remove specific memorized sequences from the model by purposefully increasing their k value in k/n-correction.

For black-box models with API access, limiting the granularity of returned logits (e.g., returning only the Top-1 token or adding calibrated noise) can effectively mitigate the attack by corrupting the high-precision membership signals required for iterative calibration. We evaluate the impact of limiting the returned logprobs to Top-H tokens in Figure 8. A sequence is recoverable if its member token is ranked in the top-H tokens during the generation process. A clear decreasing trend of recoverability is observed as H decreases, showing the effectiveness of this defense strategy. Also, sequences with larger k values are less recoverable even when $H = 20$.

7 Conclusion

Membership inference attacks and data extraction attacks have been studied to measure the privacy risk of language models. However, the set of data points that could be extracted successfully using a data extraction attack is much smaller than the set of data points whose membership can be successfully inferred. In this paper, we fill this gap by introducing membership decoding. We show that by guiding the data extraction attack using advanced membership inference techniques, we can generate training data. Another limitation of existing extraction attacks is that they only focus on generating verbatim memorization. As we show in this paper, our method enables reconstructing member data, even though it isn't fully memorized by the model in generation. Our results enable a novel direction to study LLMs' memorization of partially memorized data whose risk is unexplored in the literature.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Stella Biderman, Usvsn Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raff. 2023. Emergent and predictable memorization in large language models. *NeurIPS* 36 (2023), 28072–28090.

[3] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *ICML*. PMLR, 2397–2430.

[4] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2022. Membership inference attacks from first principles. In *2022 IEEE SP*. IEEE, 1897–1914.

[5] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2022. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*.

[6] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. 2021. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*. 2633–2650.

[7] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).

[8] Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. 2024. Do Membership Inference Attacks Work on Large Language Models?. In *Conference on Language Modeling (COLM)*.

[9] Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833* (2018).

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

[11] Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. 2024. Olmo: Accelerating the science of language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*. 15789–15809.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
[13] Jamie Hayes, Marika Swanberg, Harsh Chaudhari, Itay Yona, Ilia Shumailov, Milad Nasr, Christopher A Choquette-Choo, Katherine Lee, and A Feder Cooper. 2025. Measuring memorization in language models via probabilistic extraction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 9266–9291.
[14] Daphne Ippolito, Florian Tramèr, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher A Choquette-Choo, and Nicholas Carlini. 2022. Preventing verbatim memorization in language models gives a false sense of privacy. *arXiv preprint arXiv:2210.17546* (2022).
[15] Aly M Kassem, Omar Mahmoud, Niloofar Mireshghallah, Hyunwoo Kim, Yulia Tsvetkov, Yejin Choi, Sherif Saad, and Santu Rana. 2024. Alpaca against vicuna: Using llms to uncover memorization of llms. *arXiv preprint arXiv:2403.04801* (2024).
[16] Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schölkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. 2023. Membership inference attacks against language models via neighbourhood comparison. *arXiv preprint arXiv:2305.18462* (2023).
[17] Daniel Melcer, Nathan Fulton, Sanjay Krishna Gouda, and Haifeng Qian. 2024. Constrained Decoding for Fill-in-the-Middle Code Language Models via Efficient Left and Right Quotienting of Context-Sensitive Grammars. *arXiv preprint arXiv:2402.17988* (2024).
[18] Fatemehsadat Mireshghallah, Kartik Goyal, Archit Uniyal, Taylor Berg-Kirkpatrick, and Reza Shokri. 2022. Quantifying privacy risks of masked language models using membership inference attacks. *arXiv preprint arXiv:2203.03929* (2022).
[19] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2021. Fast model editing at scale. *arXiv preprint arXiv:2110.11309* (2021).
[20] Mustafa Safa Ozdayi, Charith Peris, Jack FitzGerald, Christophe Dupuy, Jimit Majmudar, Haidar Khan, Rahil Parikh, and Rahul Gupta. 2023. Controlling the extraction of memorized data from large language models via prompt-tuning. *arXiv preprint arXiv:2305.11759* (2023).
[21] Lorenzo Rossi, Michael Aerni, Jie Zhang, and Florian Tramèr. 2025. Membership Inference Attacks on Sequence Models. In *2025 IEEE Security and Privacy Workshops (SPW)*. IEEE, 98–110.
[22] Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. 2023. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789* (2023).
[23] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE SP*. IEEE, 3–18.
[24] Trishita Tiwari and G Edward Suh. 2024. Sequence-Level Leakage Risk of Training Data in Large Language Models. *arXiv preprint arXiv:2412.11302* (2024).
[25] Zhepeng Wang, Runxue Bao, Yawen Wu, Jackson Taylor, Cao Xiao, Feng Zheng, Weiwen Jiang, Shangqian Gao, and Yanfu Zhang. 2024. Unlocking memorization in large language models with dynamic soft prompting. *arXiv preprint arXiv:2409.13853* (2024).
[26] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 38–45.
[27] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).
[28] Roy Xie, Junlin Wang, Ruomin Huang, Minxing Zhang, Rong Ge, Jian Pei, Neil Zhenqiang Gong, and Bhuwan Dhingra. 2024. Recall: Membership inference via relative conditional log-likelihoods. *arXiv preprint arXiv:2406.15968* (2024).
[29] Jiayuan Ye, Aadyaa Maddi, Sasi Kumar Murakonda, Vincent Bindschaedler, and Reza Shokri. 2022. Enhanced membership inference attacks against machine learning models. In *Proceedings of the 2022 ACM SIGSAC conference on computer and communications security*. 3093–3106.
[30] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*. IEEE, 268–282.
[31] Weichen Yu, Tianyu Pang, Qian Liu, Chao Du, Bingyi Kang, Yan Huang, Min Lin, and Shuicheng Yan. 2023. Bag of tricks for training data extraction from language models. In *ICML*. PMLR, 40306–40320.
[32] Sajjad Zarifzadeh, Philippe Liu, and Reza Shokri. 2023. Low-cost high-power membership inference attacks. *arXiv preprint arXiv:2312.03262* (2023).
[33] Jingyang Zhang, Jingwei Sun, Eric Yeats, Yang Ouyang, Martin Kuo, Jianyi Zhang, Hao Frank Yang, and Hai Li. 2024. Min-k%++: Improved baseline for detecting pre-training data from large language models. *arXiv preprint arXiv:2404.02936* (2024).
[34] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).
[35] Weichao Zhang, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Yixing Fan, and Xueqi Cheng. 2024. Pretraining data detection for large language models: A divergence-based calibration method. *arXiv preprint arXiv:2409.14781* (2024).

A Appendix

A.1 Memorization with k/n-correction

Percent@k	0	1	2	3	4	5	6	7	8	9	10
ArXiv	2.3	7.7	12.0	13.8	17.9	17.6	16.9	7.8	2.8	1.1	0.1
DM Math	4.8	12.1	19.4	21.6	20.3	12.9	6.2	2.2	0.4	0.1	0.0
GitHub	44.2	16.6	13.2	9.0	6.5	5.1	3.3	1.0	0.8	0.3	0.0
HNews	2.7	1.3	4.5	7.6	16.5	19.9	21.5	15.3	8.6	2.0	0.1
Pile-CC	4.4	2.0	4.1	10.1	16.5	22.1	17.8	14.2	5.7	2.9	0.2
PubMed	1.3	4.0	8.3	15.2	21.4	20.3	16.6	8.4	3.3	1.0	0.2
Wiki	13.6	14.0	12.5	10.7	13.7	14.2	10.6	7.0	2.9	0.8	0.0
Average	10.5	8.2	10.6	12.6	16.1	16.0	13.3	8.0	3.5	1.2	0.1

Table 6: Detailed k-value distribution on 7 datasets on Pythia-6.9b

In this section, we present the detailed k-value distribution of Pythia-6.9b on 7 datasets in Table 6. The percentage of sequences with $k = 0$ is around 10% on average. The GitHub dataset presents the highest percentage of verbatim memorization at 44.2%, while 1.3% on PubMed dataset. The average percentage of $k = 0$ is largely biased by the GitHub dataset. The partial memorization issue is more prevalent in general.

A.2 Parameter Analysis on Calibration

We provide the detailed result of the calibration parameter α in Table 7. The performance drops to 0 when $\alpha = 1$, which underestimates the probability of IN models as the same as OUT models.

We also provide two detailed examples in computing the expected attack success rate for different α values on DM-Mathmatics and GitHub datasets with Pythia-6.9b.

On DM-Mathematics dataset and Pythia-6.9b,

$$4.8 * 81.2 + 12.1 * 6.6 + 19.4 * 0 = 469.62, \quad \alpha = 0$$
$$4.8 * 79.2 + 12.1 * 7.4 + 19.4 * 0 = 469.70, \quad \alpha = 0.25$$
$$\mathbf{4.8 * 75.0 + 12.1 * 9.1 + 19.4 * 0.5 = 479.81}, \quad \alpha = \mathbf{0.5}$$
$$4.8 * 70.8 + 12.1 * 8.3 + 19.4 * 0.5 = 449.97, \quad \alpha = 0.75$$
$$4.8 * 0 + 12.1 * 0 + 19.4 * 0 = 0, \quad \alpha = 1$$

On GitHub dataset and Pythia-6.9b,

$$44.2 * 96.2 + 16.6 * 6.0 + 13.2 * 0 = 4,351.64 \quad \alpha = 0$$
$$44.2 * 95.5 + 16.6 * 7.8 + 13.2 * 0.8 = 4,361.14 \quad \alpha = 0.25$$
$$\mathbf{44.2 * 94.1 + 16.6 * 12.0 + 13.2 * 0.8 = 4,368.98}, \quad \alpha = \mathbf{0.5}$$
$$44.2 * 89.6 + 16.6 * 16.9 + 13.2 * 1.5 = 4,260.66 \quad \alpha = 0.75$$
$$44.2 * 0 + 16.6 * 0 + 13.2 * 0 = 0 \quad \alpha = 1$$

Table 7: Calibration parameter α.

k	Method	Wikipedia					Pile-CC					ArXiv					PubMed Central				
		1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b
0	α=0	97.0	96.3	93.3	92.6	90.5	100.	100.	97.0	95.5	100.	100.	89.5	90.5	95.7	86.4	84.6	81.8	83.3	100.	93.3
	α=0.25	94.1	94.4	90.0	89.0	87.8	93.8	100.	93.9	93.2	98.0	86.7	84.2	90.5	91.3	81.8	84.6	81.8	83.3	100.	93.3
	α=0.5	86.1	84.3	80.8	83.8	79.7	93.8	90.9	90.9	86.4	93.9	73.3	68.4	71.4	78.3	72.7	76.9	81.8	83.3	92.3	93.3
	α=0.75	76.2	75.0	75.8	75.0	69.6	90.6	84.8	84.8	84.1	89.8	46.7	52.6	61.9	47.8	54.5	69.2	81.8	83.3	84.6	80.0
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	α=0	2.2	3.1	5.8	5.7	1.4	0.0	0.0	9.1	0.0	5.6	0.0	1.7	1.5	1.3	2.3	0.0	2.4	0.0	2.5	4.4
	α=0.25	3.7	4.7	6.6	6.4	2.9	0.0	0.0	9.1	10.0	5.6	1.9	3.3	3.0	1.3	3.4	0.0	2.4	0.0	5.0	4.4
	α=0.5	4.4	5.4	8.0	10.0	6.5	0.0	0.0	9.1	20.0	11.1	0.0	5.0	3.0	2.6	6.8	0.0	2.4	2.2	2.5	6.7
	α=0.75	4.4	10.9	10.9	7.9	9.4	0.0	0.0	9.1	15.0	5.6	0.0	5.0	6.0	7.8	6.8	0.0	4.9	2.2	2.5	4.4
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	α=0	0.0	0.0	0.9	0.0	0.0	2.8	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	α=0.25	0.0	0.0	0.9	0.0	0.9	2.8	0.0	0.0	2.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	α=0.5	0.0	0.0	0.0	0.0	0.9	2.8	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	α=0.75	0.0	0.0	0.9	0.8	1.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

k	Method	HackerNews					DM Mathematics					GitHub					Average				
		1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b	1b	1.4b	2.8b	6.9b	12b
0	α=0	100.	100.	100.	100.	100.	79.1	84.1	80.4	81.2	82.4	96.1	96.5	98.2	96.2	98.5	93.8	92.6	91.8	94.5	93.0
	α=0.25	100.	100.	100.	100.	100.	74.4	79.5	78.4	79.2	78.4	95.0	95.7	97.2	95.5	97.3	89.8	90.8	90.5	92.6	91.0
	α=0.5	100.	100.	100.	100.	100.	65.1	68.2	76.5	75.0	76.5	92.3	93.8	94.5	94.1	96.2	83.9	83.9	85.3	87.1	87.5
	α=0.75	96.0	96.0	96.6	96.3	96.7	55.8	61.4	68.6	70.8	70.6	86.6	90.0	90.2	89.6	91.4	74.5	77.4	80.2	78.3	78.9
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.4
1	α=0	9.1	8.3	0.0	7.7	0.0	5.2	4.8	3.8	6.6	5.8	8.8	7.9	10.0	6.0	4.2	3.6	4.0	4.3	4.3	3.4
	α=0.25	9.1	8.3	0.0	7.7	0.0	7.3	4.8	5.7	7.4	7.5	11.0	10.9	13.7	7.8	6.2	4.7	4.9	5.4	6.5	4.3
	α=0.5	9.1	8.3	0.0	7.7	0.0	7.3	7.7	7.5	9.1	9.2	13.8	17.0	16.3	12.0	10.4	4.9	6.6	6.6	9.1	7.2
	α=0.75	18.2	0.0	12.5	7.7	0.0	7.3	8.7	6.6	8.3	10.0	17.1	18.2	18.4	16.9	15.6	6.7	6.8	9.4	9.4	7.4
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	α=0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5	0.0	0.0	0.9	0.0	0.0	0.5	0.1	0.2	0.0	0.1
	α=0.25	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.5	0.8	0.0	2.6	0.8	0.0	0.7	0.1	0.6	0.5	0.2
	α=0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.5	1.1	1.6	0.8	3.5	0.8	1.0	0.8	0.2	0.6	0.2	0.4
	α=0.75	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.5	1.1	3.2	1.6	2.6	1.5	3.9	0.5	0.3	0.6	0.5	1.0
	α=1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0