A Comprehensive Analysis of Memorization in Large Language Models

Hirokazu Kiyomaru[1*] **Issa Sugiura**[2*] **Daisuke Kawahara**[1,3] **Sadao Kurohashi**[1,2]

[1]Research and Development Center for LLMs, National Institute of Informatics
[2]Kyoto University [3]Waseda University

kiyomaru@nii.ac.jp sugiura.issa.q29@kyoto-u.jp
dkw@waseda.jp kurohashi@nii.ac.jp

Abstract

This paper presents a comprehensive study that investigates memorization in large language models (LLMs) from multiple perspectives. Experiments are conducted with the Pythia and LLM-jp model suites, both of which offer LLMs with over 10B parameters and full access to their pre-training corpora. Our findings include: (1) memorization is more likely to occur with larger model sizes, longer prompt lengths, and frequent texts, which aligns with findings in previous studies; (2) memorization is less likely to occur for texts not trained during the latter stages of training, even if they frequently appear in the training corpus; (3) the standard methodology for judging memorization can yield false positives, and texts that are infrequent yet flagged as memorized typically result from causes other than true memorization[1].

1 Introduction

Large language models (LLMs) have revolutionized the field of natural language processing by demonstrating an impressive ability to generate coherent text, perform complex language understanding tasks, and store a wealth of real-world knowledge (Brown et al., 2020). The impact of LLMs is spreading across society, and their uses are increasingly explored in various applications (Kaddour et al., 2023).

However, LLMs still have many concerns; *memorization* is one of them. LLMs are known to memorize portions of their training corpora (Carlini et al., 2021). Memorization can cause crucial issues, including unintentional reproduction of copyrighted materials (Lee et al., 2023) and personal information (Huang et al., 2022). Understanding the extent and nature of memorization is essential for developing secure and reliable LLMs.

[*]Equal contributions
[1]Our code is available at https://github.com/llm-jp/memorization-analysis.

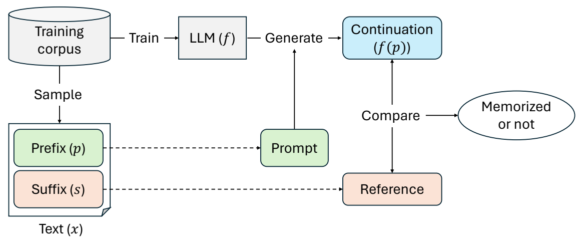


Figure 1: Overview of the standard methodology for investigating memorization in LLMs quantitatively. Text x in the training corpus is split into the prefix p and the suffix s. Given p, the LLM f generates the continuation $f(p)$. If $f(p)$ matches or closely resembles the suffix s, s is considered memorized in the LLM.

This study comprehensively evaluates memorization in LLMs, integrating multiple definitions of memorization and key factors contributing to memorization, which are discussed separately in different literature.

We follow the standard methodology for quantitatively investigating memorization in LLMs, as illustrated in Figure 1. In this methodology, an LLM is given a prompt and generates the continuation. Memorization is identified by checking if the continuation replicates text from the training corpus.

We explore two memorization types: verbatim memorization (Carlini et al., 2021) and approximate memorization (Ippolito et al., 2023). Verbatim memorization refers to the exact reproduction of text from the training corpus, while approximate memorization allows for slight variations. We examine these memorization types through the size of model parameters (Tirumala et al., 2022; Carlini et al., 2023; Ishihara, 2024), the length of prompts (Carlini et al., 2023; Ishihara, 2024), the duplication counts of text in the training corpus (Carlini et al., 2023; Ishihara, 2024), and the training step at which text is trained (Tirumala et al., 2022; Jagielski et al., 2023).

We conduct experiments using fully open LLMs: the Pythia model suite (Biderman et al., 2023) and the LLM-jp v1.0 model suite (LLM-jp, 2024). The Pythia model suite offers LLMs of various parameter sizes, from 14M to 12B parameters, trained on an English corpus, whereas the LLM-jp v1.0 model suite has two LLMs with 1.3B and 13B parameters, primarily trained on a mix of English and Japanese corpora. Both model suites are released with their pre-training corpora, allowing for analysis of memorization.

Our key findings are three-fold:

- Memorization is more likely to occur with larger model sizes, longer prompt lengths, and frequent texts across different memorization definitions and model suites.
- Memorization is less likely to occur for texts not included in the latter stages of training, even if they are frequent.
- The standard methodology for judging memorization can yield false positives, and texts that are infrequent yet flagged as memorized typically result from other factors, such as duplication of the prompt, rather than true memorization.

2 Related Work

Once memorization in LLMs was first identified by Carlini et al. (2021), it has been explored from various perspectives.

A line of work studies methods to better extract memorized texts from LLMs, making a research subfield called training data extraction attack (Ishihara, 2023). Most existing methods follow the methodology proposed in Carlini et al. (2021) consisting of two steps: candidate generation and membership inference (Ishihara, 2023; Nasr et al., 2023).

Another line of work investigates the causes and mechanisms of memorization. Carlini et al. (2023) found that verbatim memorization is more likely to happen with larger model sizes, longer prompt lengths, and frequent texts. Tirumala et al. (2022) focused on analyzing the dynamics of memorization and found that larger models memorize their training corpora more quickly. Tirumala et al. (2022) also investigated how language models forget memorized texts throughout training. A similar analysis was conducted by Jagielski et al. (2023).

A further line of work aims to reduce memorization to address security and privacy issues. Lee et al. (2022) and Kandpal et al. (2022) showed that deduplication of training corpora effectively reduces memorization without hurting the performance in downstream tasks. Ippolito et al. (2023) proposed a decoding method named MemFree decoding, which is guaranteed to eliminate verbatim memorization by preventing the generation of n-grams present in the training corpus. Ippolito et al. (2023) also showed that while MemFree decoding perfectly prevents verbatim memorization, LLMs still generate texts that closely resemble those in their training corpora. This phenomenon is termed approximate memorization.

As for the LLMs to explore, most previous studies use monolingual LLMs trained on public English corpora, such as GPT-Neo (Black et al., 2022) and Pythia (Biderman et al., 2023), with some exceptions such as Ishihara (2024), who trains a Japanese language model on an in-house, domain-specific corpus.

Our study incorporates insights from previous studies and presents a comprehensive analysis of memorization. Besides, our analysis utilizes not only a monolingual LLM primarily trained on an English corpus but also a multilingual LLM trained on a mix of English and Japanese corpora.

3 Methodology

This section describes our methodology to comprehensively investigate memorization in LLMs. Our analysis integrates multiple definitions of memorization and key factors contributing to memorization, which are discussed separately in previous studies.

3.1 Definitions of Memorization

We start by defining memorization. Figure 1 shows the standard procedure for investigating memorization in LLMs, to which we adhere.

Notation We investigate the memorization of an auto-regressive language model f. Let x be a sequence of consecutive tokens with a length of ℓ in the training corpus. We split x into the prefix p and the suffix s, so $x = [p \parallel s]$. The prefix p is used to prompt the model f to generate the continuation $f(p)$.

Verbatim memorization (Carlini et al., 2023) The suffix s is considered verbatim memorized if s is identical to $f(p)$.

Tokenizer	Example	Near-Duplicate Example	J_W
Pythia	**\n \n **New England** Aka Hairy Duskywing \n Male, dorsal \n **RECOGNITION** < 1.5 in. The usual duskywing pattern of alternating black and buff patches against	URGESS) 1870**\n \n **"New England"** Aka Aspen Duskywing \n Male, dorsal \n **RECOGNITION** < 1.5 in. Small for a duskywing	0.613
LLM-jp v1.0	駐車場共用 「春日 食堂 イオン大野城 店」の運営者様・オーナー様は 食べログ店舗準会員（無料）にご登録ください。ご登録はこちら春日 食堂 イオン大野城店 09 2-5	-1博 多 南 駅 か ら451m 「 黒 田 屋春日店」の運営者様・オーナー様は食べログ店舗準会員（無料）にご登録ください。ご登録はこちら黒田屋 春日店 09	0.612

Table 1: Text pairs with weighted Jaccard indexes close to 0.6. Overlaps are highlighted in yellow.

Algorithm 1 Fast Near-duplicate Matching

Input: Suffix s, document d, and n of n-gram
Output: Whether d has a span near-duplicate to s

```
 1: $\ell_s \leftarrow \text{len}(s)$
 2: $\ell_d \leftarrow \text{len}(d)$
 3: $H \leftarrow \text{HashSet}(\text{Ngram}(s, n))$
 4: $\delta \leftarrow 0.6$
 5: for $i = 0$ to $\max(\ell_d - \ell_s, 0)$ do
 6:     if $d[i : i + n] \in H$ then
 7:         for $j = \max(i - \ell_s + n, 0)$ to $i$ do
 8:             $t \leftarrow d[j : j + \ell_s]$
 9:             if $J_W(s, t) \geq \delta$ then
10:                 return True
11:             end if
12:         end for
13:     end if
14: end for
15: return False
```

Approximate memorization (Ippolito et al., 2023) The suffix s is recognized as approximately memorized if the BLEU score (Papineni et al., 2002) between s and $f(p)$ exceeds a certain threshold. Following Ippolito et al. (2023), we adopt a threshold of 0.75 throughout the paper.

3.2 Factors to Explore

Previous studies identify several factors that contribute to memorization. This study examines if such factors remain consistent across different definitions of memorization and varying model suites.

Parameter size Previous studies suggest that LLMs with larger parameter sizes memorize more data (Carlini et al., 2021; Tirumala et al., 2022; Carlini et al., 2023; Ishihara, 2024). To examine this factor, we use model suites that provide LLMs with different parameter sizes.

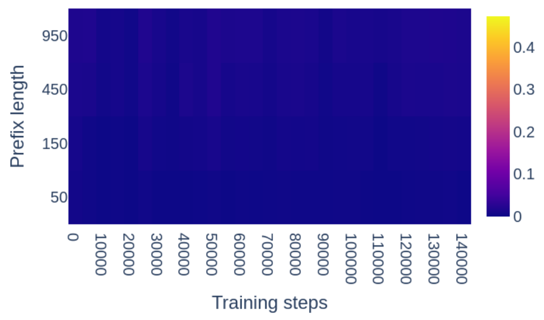
Context length It is suggested that memorization is more likely to occur as the length of the prompts increases (Carlini et al., 2023; Ishihara, 2024). We examine this factor by varying the length of prefixes $|p|$.

Duplication Count The duplication count of text in the training corpus is known to be an influential factor in memorization (Kandpal et al., 2022; Carlini et al., 2023; Ishihara, 2024). We investigate this factor by grouping suffixes s according to their duplication counts, measured as the number of documents containing s in the training corpus.

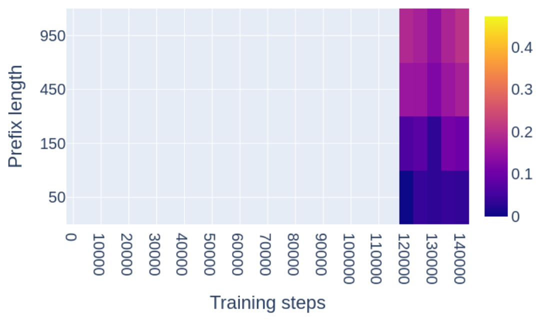
We explore two ways to count duplicates. First, we count the number of documents in the training corpus that contain the text identical to s, which we refer to as the **exact duplication count**. Most previous studies count duplication counts in this manner (Carlini et al., 2023; Ishihara, 2024). In addition, we count the number of documents containing near-duplicate texts to s, which we refer to as the **near-duplication count**. The method for obtaining near-duplication counts is detailed in Section 3.3.

Training step Tirumala et al. (2022) and Jagielski et al. (2023) analyze how training steps at which text is trained affect its memorization. We explore this factor by identifying the last training step at which the suffix s is trained.

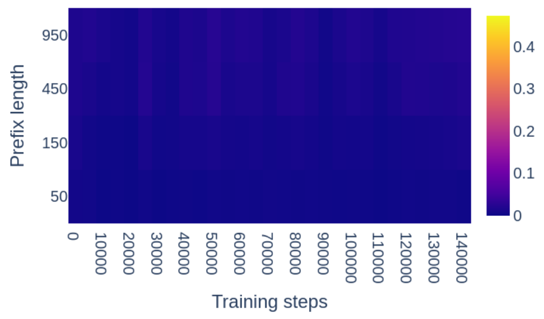
It is important to note that previous studies train small models with approximately 100M parameters to examine this factor (Tirumala et al., 2022; Jagielski et al., 2023). Conversely, this study uses



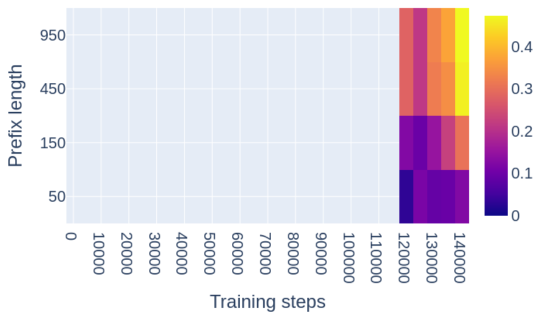
(a) Pythia 1.4B's verbatim memorization for text with exact duplication counts of 1 to 10.



(b) Pythia 1.4B's verbatim memorization for text with exact duplication counts of 11 to 100.



(c) Pythia 12B's verbatim memorization for text with exact duplication counts of 1 to 10.



(d) Pythia 12B's verbatim memorization for text with exact duplication counts of 11 to 100.

Figure 2: Verbatim memorization of the Pythia model suite. The x-axis represents the last-seen training steps of suffixes s. The y-axis represents the lengths of prefixes p. The brightness shows the fraction of examples recognized as verbatim memorization. Blank grids indicate that there were fewer than 10 examples, failing to provide meaningful statistics.

LLMs with more than 10B parameters for the analysis, which are much closer to those used in practical scenarios.

3.3 Near-duplication Count

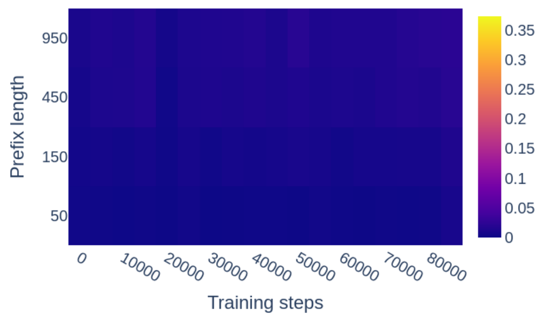
Aiming to investigate the memorization of truly infrequent and unique text, we conduct an analysis based on near-duplication counts. As Section 4.6 will demonstrate, some texts with small exact duplication counts are approximately memorized, but they often have numerous near-duplicate counterparts in the training corpus. We disentangle such text from genuinely infrequent text by counting near-duplicate matches for an in-depth analysis of memorization in infrequent text.

To this end, we count the number of documents containing near-duplicate text for each suffix s. We use the weighted Jaccard similarity to judge if a text pair is near-duplicate. The weighted Jaccard similarity is an extension of the Jaccard similarity to consider the duplication of elements. We consider a text as a multiset of tokens and apply the weighted Jaccard similarity as follows:

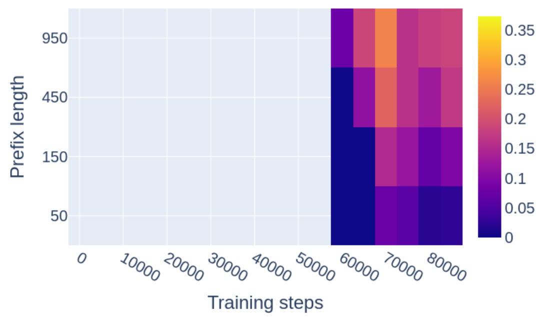
$$J_W(\boldsymbol{a}, \boldsymbol{b}) := \frac{\sum_i \min(a_i, b_i)}{\sum_i \max(a_i, b_i)}, \qquad (1)$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are frequency vectors in which i-th element corresponds to the frequency of the i-th token in the vocabulary. We regard text pairs with a weighted Jaccard similarity of 0.6 or higher as near-duplicate. The threshold is determined based on a qualitative inspection. Table 1 shows examples of text pairs close to this threshold.

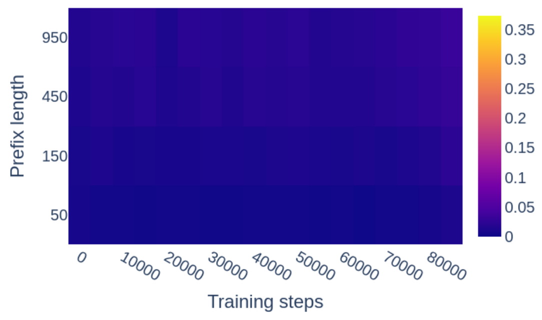
Due to the huge size of the training corpus, computing similarities between all text spans and all suffixes is infeasible. Therefore, we propose a fast algorithm based on the Rabin-Karp algorithm (Karp and Rabin, 1987). Algorithm 1 shows



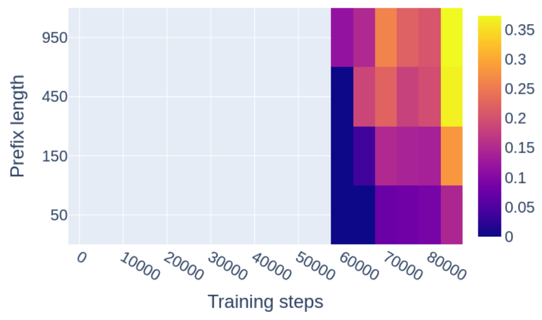
(a) Verbatim memorization in LLM-jp 1.3B for text with exact duplication counts of 1 to 10.



(b) Verbatim memorization in LLM-jp 1.3B for text with exact duplication counts of 11 to 100.



(c) Verbatim memorization in LLM-jp 13B for text with exact duplication counts of 1 to 10.



(d) Verbatim memorization in LLM-jp 13B for text with exact duplication counts of 11 to 100.

Figure 3: Verbatim memorization of the LLM-jp model suite.

the procedure. First, we set the length of text spans to be the same as s. Besides, we filter out text spans with no shared n-grams as s. This is a natural constraint that holds for text pairs that appear to duplicate qualitatively, and the check can be quickly done by making the hash set of the n-grams in s in advance. In this study, we employ $n = 10$. For text spans containing any of the n-grams in s, we calculate the weighted Jaccard similarity and recognize them as near-duplicate if the similarity exceeds the threshold. A detailed analysis of this algorithm, including a discussions on its computational cost, can be found in Appendix A.

4 Experiments

We conducted experiments to investigate memorization defined in Section 3.1 from the perspectives discussed in Section 3.2.

4.1 Models

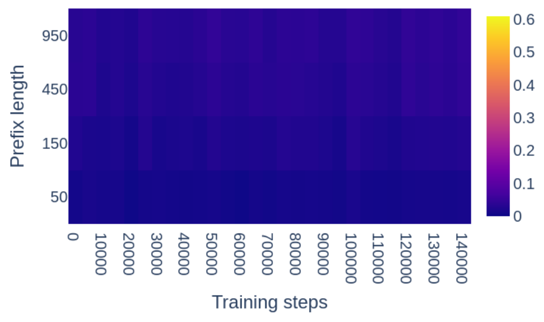
We used the Pythia and LLM-jp model suites. Both model suites offer LLMs with varying parameters and provide access to their pre-training corpora.

Pythia Pythia (Biderman et al., 2023) is a suite of LLMs trained on a public English corpus, the Pile dataset (Gao et al., 2020; Biderman et al., 2022), containing 300B tokens. We used the Pythia models with 1.4B and 12B parameters in our experiments.

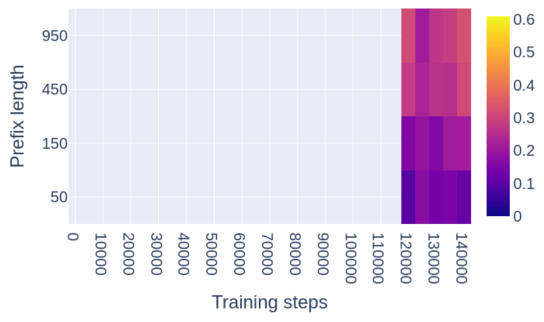
LLM-jp LLM-jp v1.0 (LLM-jp, 2024) is a suite of LLMs trained primarily on a mix of Japanese and English corpora with 270B tokens in total. As for the Japanese corpus, LLM-jp v1.0 uses Japanese Wikipedia and the Japanese portion of the multilingual C4 dataset (Raffel et al., 2020). As for the English corpus, English Wikipedia and the Pile dataset are used. We used the LLM-jp v1.0 models with 1.3B and 13B parameters in our experiments.

4.2 Evaluation Data

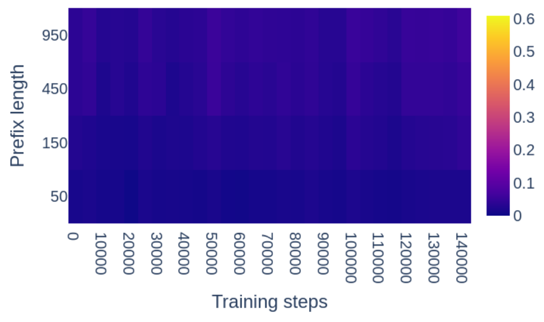
For each model suite, we randomly sampled approximately 30,000 sequences of consecutive tokens of length 50 from the training corpus as suf-



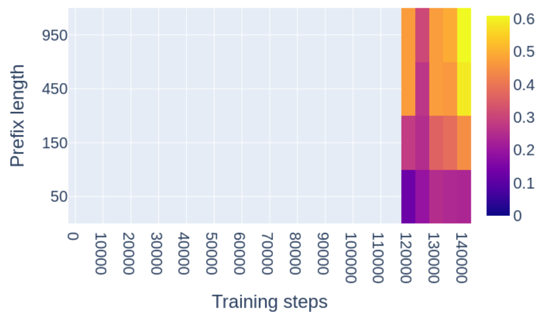
(a) Approximate memorization in Pythia 1.4B for text with exact duplication counts of 1 to 10.



(b) Approximate memorization in Pythia 1.4B for text with exact duplication counts of 11 to 100.



(c) Approximate memorization in Pythia 12B for text with exact duplication counts of 1 to 10.



(d) Approximate memorization in Pythia 12B for text with exact duplication counts of 11 to 100.

Figure 4: Approximate memorization of the Pythia model suite.

fixes s. We then extracted their preceding tokens as prefixes p so that the total length of the concatenation of p and s (termed ℓ) equaled to $\{100, 200, 500, 1000\}$. As for the lengths to explore, we followed Carlini et al. (2023).

4.3 Implementation Details

Exact duplication count To obtain exact duplication counts, we constructed a full-text search index using ElasticSearch[2]. For each suffix s, we issued a phrase match query to count the number of documents containing s. To make a search index for each corpus with approximately 300B tokens, it took about 5 hours using an Ubuntu machine equipped with 128 CPUs and 190GB of RAM.

Near-duplication count We implemented the algorithm described in Section 3.3 in Rust. We constructed hash sets using the FxHash library[3], a fast hash implementation. We chose $n = 10$ to perform n-gram-based filtering. It took about 1.5 days to process each corpus using Ubuntu machines with 640 CPUs in total.

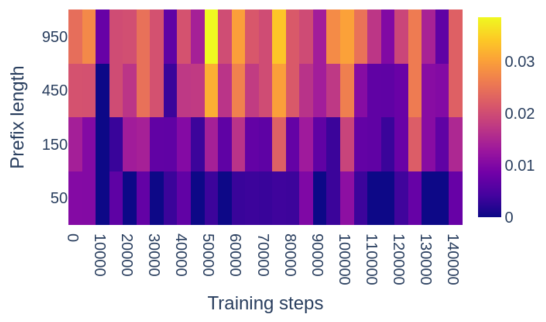
Training step To identify the last training step at which each suffix s is seen, we reused the search index constructed to obtain exact duplication counts. We issued a phrase match query for each s and obtained the largest training step from the results.

Decoding Following Carlini et al. (2023), we performed greedy decoding to generate continuations from prefixes p with models f. We forced the models to generate 50 tokens so that the lengths of generated continuations equaled the length of s, even if the models generated the EOS (end of sequence) special token. We used an Ubuntu machine equipped with 2 NVIDIA A100 40GB GPUs for this process. We used the Hugging Face transformers (Wolf et al., 2020) library to run LLMs. The total time required for generating continuations for all prefixes was approximately 3 hours.

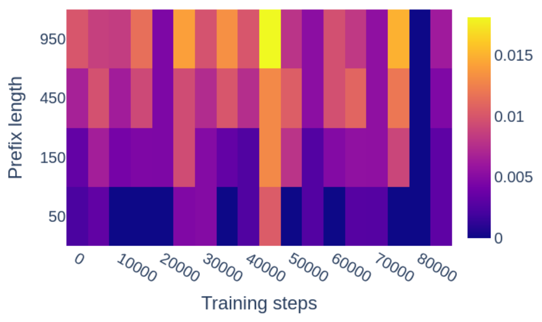
[2] `https://www.elastic.co/`
[3] `https://github.com/cbreeden/fxhash`

Model	Approximately memorized text	Near-duplicate counterpart in the corpus
Pythia 12B	dx21 < q) {\n info = -12;\n LAPACKE_xerbla("LAPACKE_dorbdb_work", info);\n return info;\n }\n if(	ldvt < ncols_vt) {\n info = -18;\n LAPACKE_xerbla("LAPACKE_cgesvdx_work", info);\n return info; }
LLM-jp 13B	バラ場合での査定か無料にて、お客様の切手を査定するスタッフの顔写真も。 越中島駅 切手 買取り1シートから、たった一枚で普通切手、お休みが異なる場合がございます。どちらも	バラ場合での査定か無料にて、お客様の切手を査定するスタッフの顔写真も。 ささしまライブ駅 切手 買取り1シートから、たった一枚で普通切手、お休みが異なる場合がございます。どちらも

Table 2: Examples of approximately memorized texts and their near-duplicate counterparts in the training corpus. Overlaps are highlighted in yellow.



(a) Approximate memorization in Pythia 12B for text with no near-duplicate.



(b) Approximate memorization in LLM-jp 13B for text with no near-duplicate.

Figure 5: Approximate memorization of the Pythia 12B and LLM-jp 13B models for text with no near-duplicate. Note that the maximum memorization ratio in this figure is much lower than that in Figure 4, indicating that memorization rarely occurs for texts having no near-duplicates.

4.4 Impact of Model Size, Context Length, Training Step, and Exact Duplication Count on Verbatim Memorization

Figures 2 and 3 show the ratio of verbatim memorization of the Pythia and LLM-jp model suites, respectively. Both model suites exhibit similar tendencies. That is, memorization is more likely to occur with larger model sizes, longer context lengths, and larger duplication counts, which aligns with the findings in Carlini et al. (2023). Besides, memorization is less likely to occur for texts not included in the final stages of training, even if they are frequent.

4.5 Impact of Model Size, Context Length, Training Step, and Exact Duplication Count on Approximate Memorization

We performed the same analysis for approximate memorization. Figure 4 shows the ratio of approximate memorization of the Pythia model suite. Compared to verbatim memorization, the ratio of approximate memorization is much larger. Specifically, we observed a maximum ratio of about 0.4 for verbatim memorization and about 0.6 for approximate memorization. However, we still found the consistent contributions of model sizes, context lengths, training steps, and exact duplication counts to memorization. We confirmed the same tendencies for the LLM-jp model suite.

Type	Model	Prefix	Suffix
Copy from prefix	Pythia	[...] consider yourself a right-winger and yet you're quoting a Trotskyist left-winger. Trotskyist who turned neocon, just like so many (Kristol, Perle, Wolfowitz in the USA but there also are a lot [...] consider yourself a	right-winger and yet you're quoting a Trotskyist left-winger. Trotskyist who turned neocon, just like so many (Kristol, Perle, Wolfowitz in the USA but there also are a lot
Regular pattern	LLM-jp	[...] 5 巻 – 蒐集匣柴田昌弘『紅い牙 ブルー・ソネット』 6 巻 – 蒐集匣柴田昌弘『紅い牙 ブルー	・ソネット』 7 巻 – 蒐集匣柴田昌弘『紅い牙 ブルー・ソネット』 8 巻 – 蒐集匣柴田昌弘『紅い牙 ブルー・ソネット』

Table 3: Examples of approximate memorization occurred in texts with no near-duplicates. Overlaps are highlighted in yellow. Red highlights show the parts that follow a regular pattern. The symbol "[...]" indicates omission.

Type	Pythia 12B	LLM-jp 13B
Copy from prefix	55%	60%
Regular pattern	45%	40%
Memorization		
w/ near-duplicates	0%	20%
w/o near-duplicates	0%	0%

Table 4: The plausible reasons to be recognized as approximately memorized and their ratios for texts without near-duplicates in the training corpus. The sum of the ratios may not necessarily equal one because multiple reasons can be combined in single examples.

4.6 Qualitative Analysis of Memorization in Text with Low Exact Duplication Count

Texts with low exact duplication counts were rarely memorized, but it does occur. What kind of texts do LLMs memorize after seeing them only once?

One of the authors manually investigated the characteristics of such texts and found that most of them had numerous near-duplicate counterparts in the training corpus. Table 2 shows typical examples found in the Pythia 12B model and LLM-jp 13B model, which were identified as approximately memorized despite having no exact duplicates in the training corpus. As shown in Table 2, typical cases include texts like code snippets with different variable names and real estate advertisements with different city names. When taking near-duplicates into account, these texts are considered frequent, casting doubt on concluding that the LLMs memorized them after a single exposure.

4.7 Approximate Memorization in Text without Near-duplicates

On top of the analysis in Section 4.6, we conducted an analysis based on the near-duplication count of text to investigate if LLMs memorize unique texts after a single exposure.

Figure 5 shows the approximate memorization of the Pythia 12B and LLM-jp 13B models for texts that had no near-duplicates in the training corpus. The low maximum memorization ratio indicates that memorization rarely occurs with such texts. However, the presence of non-zero grids suggests that texts without any near-duplicates in the training corpus can still be flagged as approximately memorized.

We again conducted a manual investigation to explore the characteristics of the memorized texts, focusing on memorization that happened with prefixes with a length of 950. One of the authors manually examined 20 memorized examples for each of the Pythia 12B and LLM-jp 13B models.

Table 4 shows the plausible reasons for being flagged as approximately memorized and their ratios, with Table 3 showing the examples. Most of the memorized texts appeared to copy their prompts or exploit the regularity in the prompts to generate the continuation. In the examples from LLM-jp 13B, there were texts that seemed memorized by the model. However, we found that all such texts had near-duplicates in the training corpus. For instance, real estate advertisements with very long place names were recognized as having no near-duplicates by our algorithm based on token-level overlaps, but there are many texts in the training corpus following the same template.

In both models, we found no texts that could be attributed to genuine memorization from a single data exposure.

5 Conclusion

This paper investigated the memorization of LLMs from multiple perspectives and presented a comprehensive analysis. Our experiments confirmed that findings in previous studies are consistent across different memorization definitions and model series. Besides, our manual investigation suggested that the standard methodology for judging memorization can yield false positives, and texts that are infrequent yet flagged as memorized mostly arise from causes other than true memorization.

A crucial future work is to investigate memorization in production-grade LLMs. Although the LLMs used in our experiments represent the largest fully open LLMs, they significantly underperform when compared to production-grade LLMs, such as GPT-4 (OpenAI, 2024). The memorization of advanced models remains largely unexplored, yet it is crucial for ensuring the security and reliability of LLM applications, given their profound societal impact. We are in the process of developing a fully open LLM with 172B parameters, which will facilitate further exploration into memorization dynamics in state-of-the-art models. We plan to investigate whether our findings in this study still hold true in the model.

References

Stella Biderman, Kieran Bicheno, and Leo Gao. 2022. Datasheet for the Pile. *Preprint*, arXiv:2201.07311.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*. JMLR.org.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2023. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*.

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)*, pages 2633–2650.

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. *Preprint*, arXiv:2101.00027.

Jie Huang, Hanyin Shao, and Kevin Chen-Chuan Chang. 2022. Are large pre-trained language models leaking your personal information? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2038–2047, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Daphne Ippolito, Florian Tramer, Milad Nasr, Chiyuan Zhang, Matthew Jagielski, Katherine Lee, Christopher Choquette Choo, and Nicholas Carlini. 2023. Preventing generation of verbatim memorization in language models gives a false sense of privacy. In *Proceedings of the 16th International Natural Language Generation Conference*, pages 28–53, Prague, Czechia. Association for Computational Linguistics.

Shotaro Ishihara. 2023. Training data extraction from pre-trained language models: A survey. In *Proceedings of the 3rd Workshop on Trustworthy Natural Language Processing (TrustNLP 2023)*, pages 260–275, Toronto, Canada. Association for Computational Linguistics.

Shotaro Ishihara. 2024. Quantifying memorization of domain-specific pre-trained language models using japanese newspaper and paywalls. *Preprint*, arXiv:2404.17143.

Matthew Jagielski, Om Thakkar, Florian Tramer, Daphne Ippolito, Katherine Lee, Nicholas Carlini, Eric Wallace, Shuang Song, Abhradeep Guha Thakurta, Nicolas Papernot, and Chiyuan Zhang. 2023. Measuring forgetting of memorized training examples. In *The Eleventh International Conference on Learning Representations*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *Preprint*, arXiv:2307.10169.

Nikhil Kandpal, Eric Wallace, and Colin Raffel. 2022. Deduplicating training data mitigates privacy risks in language models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 10697–10707. PMLR.

Richard M. Karp and Michael O. Rabin. 1987. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260.

Jooyoung Lee, Thai Le, Jinghui Chen, and Dongwon Lee. 2023. Do language models plagiarize? In *Proceedings of the ACM Web Conference 2023*, pages 3637–3647, New York, NY, USA. Association for Computing Machinery.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.

LLM-jp. 2024. Llm-jp: A cross-organizational project for the research and development of fully open japanese llms. *Preprint*, arXiv:2407.03963.

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

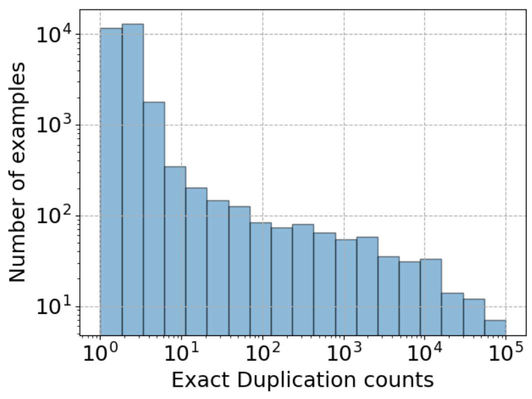
OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

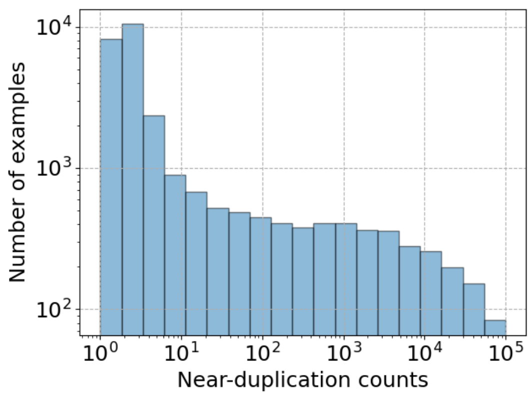
Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. 21(1).

Kushal Tirumala, Aram Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. 2022. Memorization without overfitting: Analyzing the training dynamics of large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 38274–38290. Curran Associates, Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.



(a) Exact duplication count.



(b) Near duplication count.

Figure 6: A histogram of duplication counts in the LLM-jp corpus.

A Details on the Fast Near-duplicate Matching Algorithm (Algorithm 1)

A.1 Computational Analysis

Let ℓ_s be the number of tokens in a suffix s and n be the number of n-gram. The computational complexity to calculate the hash set H of the n-grams in s is $O(n\ell_s)$, which is negligible. Calculating the weighted Jaccard index J_W between a suffix s and a text span t has a complexity of $O(|s| + |t|)$. Given that $|s| = |t| = \ell_s$ in our scenario, the complexity simplifies to $O(\ell_s)$.

Let ℓ_d denote the number of tokens in a document d and p denote the probability that a given n-gram from the document d exists in the hash set H, i.e., $d[i+n] \in H$. Using a rolling hash reduces the complexity of computing hash values for successive n-grams to $O(1)$ after the initial calculation. Hence, the total complexity of our algorithm when using a rolling hash is $O(\ell_d(1 + p\ell_d\ell_s))$. If a standard hash function with a complexity of $O(n)$ per operation is used instead, the overall complexity becomes $O(\ell_d(n + p\ell_d\ell_s))$. Given that p is typically low, the algorithm approaches linear time performance.

A.2 Choice of Hash Function

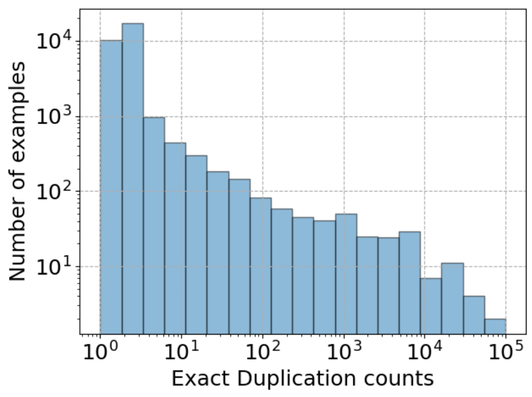
Though a rolling hash can compute the hash value of n-length tokens in $O(1)$ time using the previous hash value, it relies on computationally expensive operations (i.e., modulo). In contrast, the fxhash library offers a very fast implementation of a standard hash, and the use of a standard hash is acceptable for small values of n. Therefore, we used the fxhash library in our implementation. The code of our algorithm is available at `https://github.com/speed1313/fast-near-duplicate-matching`.

A.3 Distribution of Duplication Counts

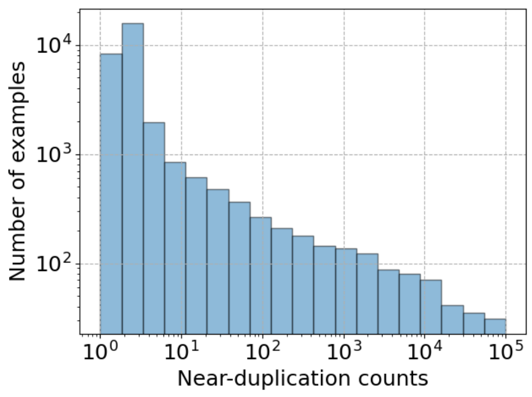
The distributions of duplication counts calculated on the LLM-jp and Pythia corpora are shown in Figure 6 and 7, respectively. For each corpus, we randomly sampled approximately 30,000 sequences of consecutive tokens of length 50 and then obtained their duplication counts.

B Models Memorize More as Duplication Counts and Prefix Lengths Scale

Figures 8, 9, 10, and 11 show the memorization of the LLM-jp and Pythia model suites, where the models memorize more as duplication counts and prefix lengths scale.

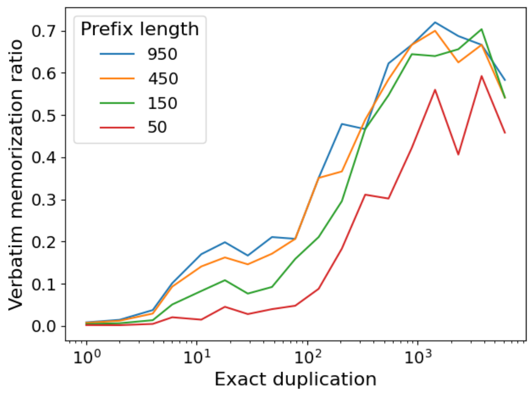


(a) Exact duplication count.

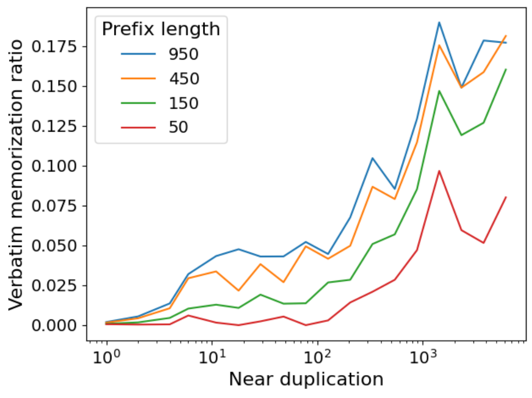


(b) Near duplication count.

Figure 7: A histogram of duplication counts in the Pile.

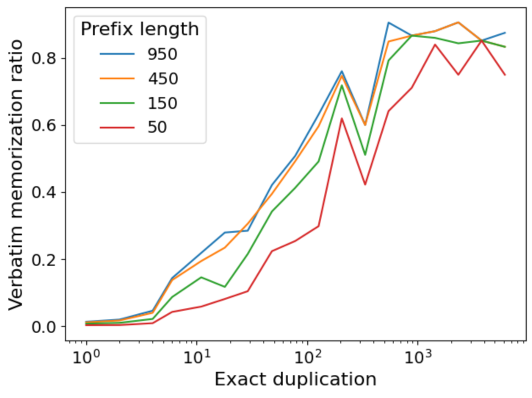


(a) Exact duplication count vs. Verbatim memorization

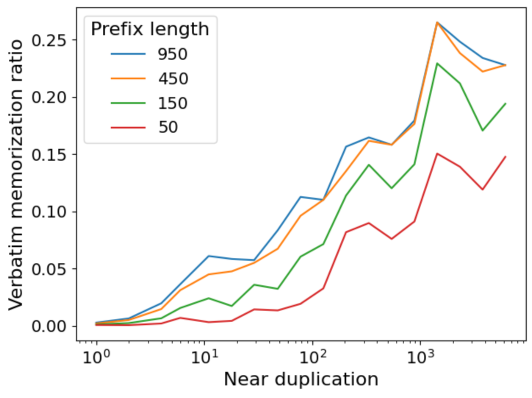


(b) Near-duplication count vs. Verbatim memorization

Figure 8: Memorization ratios in LLM-jp 1.3B.

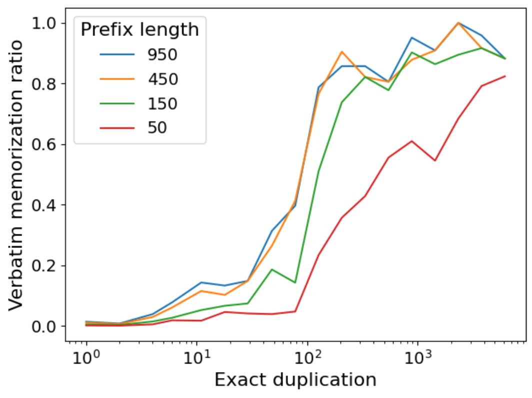


(a) Exact duplication count vs. Verbatim memorization

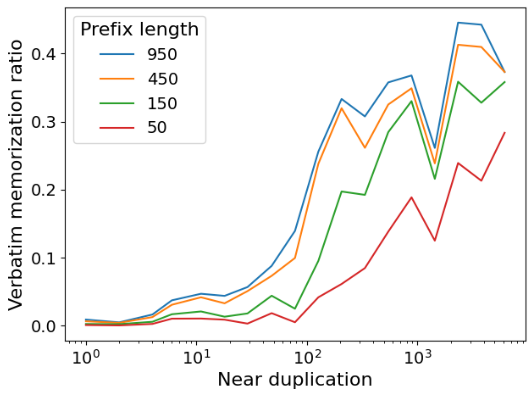


(b) Near-duplication count vs. Verbatim memorization

Figure 9: Memorization ratios in LLM-jp 13B.

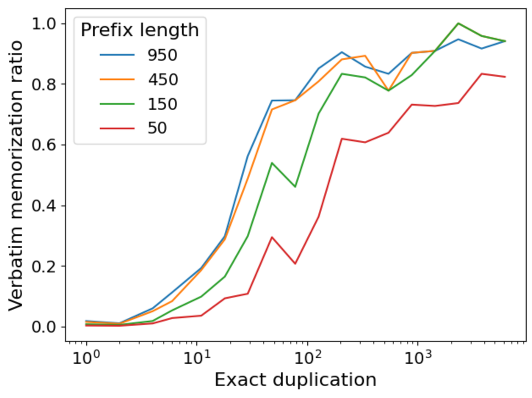


(a) Exact duplication count vs. Verbatim memorization

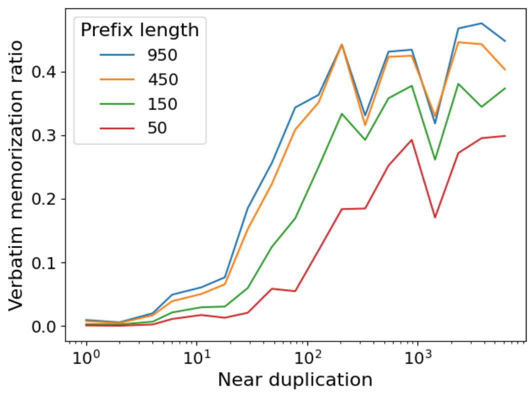


(b) Near-duplication count vs. Verbatim memorization

Figure 10: Memorization ratios in Pythia 1.4B.



(a) Exact duplication count vs. Verbatim memorization



(b) Near-duplication count vs. Verbatim memorization

Figure 11: Memorization ratios in Pythia 12B.