Demystifying Verbatim Memorization in Large Language Models

Jing Huang, Diyi Yang*, Christopher Potts*
Stanford University
`{hij, diyiy, cgpotts}@stanford.edu`

Abstract

Large Language Models (LLMs) frequently memorize long sequences verbatim, often with serious legal and privacy implications. Much prior work has studied such verbatim memorization using observational data. To complement such work, we develop a framework to study verbatim memorization in a controlled setting by continuing pre-training from Pythia checkpoints with injected sequences. We find that (1) non-trivial amounts of repetition are necessary for verbatim memorization to happen; (2) later (and presumably better) checkpoints are more likely to verbatim memorize sequences, even for out-of-distribution sequences; (3) the generation of memorized sequences is triggered by distributed model states that encode high-level features and makes important use of general language modeling capabilities. Guided by these insights, we develop stress tests to evaluate unlearning methods and find they often fail to remove the verbatim memorized information, while also degrading the LM. Overall, these findings challenge the hypothesis that verbatim memorization stems from specific model weights or mechanisms. Rather, verbatim memorization is intertwined with the LM's general capabilities and thus will be very difficult to isolate and suppress without degrading model quality.

1 Introduction

Verbatim memorization refers to LLMs outputting long sequences of texts that are exact matches of training examples (Carlini et al., 2021, 2023). Unlike recalling factual knowledge or fixed expressions, verbatim memorization can have serious copyright and privacy implications (Karamolegkou et al., 2023; Chen et al., 2024c; Lee et al., 2023; Carlini et al., 2021; Shokri et al., 2017) and potentially waste model capacity (Nasr et al., 2023). Recent work has identified data frequency and model size as factors contributing to verbatim memorization in LLMs (Carlini et al., 2023; Prashanth et al., 2024; Karamolegkou et al., 2023). However, it is still not well understood why and how LLMs verbatim memorize certain texts in training data.

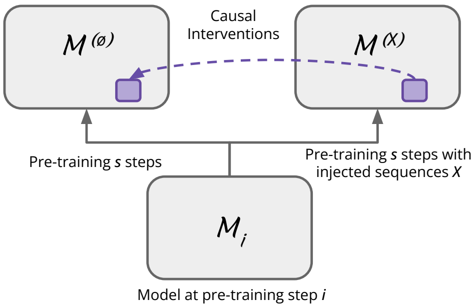

$M^{(\varnothing)}$
Causal Interventions
$M^{(X)}$
Pre-training *s* steps
Pre-training *s* steps with injected sequences *X*
M_i
Model at pre-training step *i*

Figure 1: An overview of our sequence injection framework, which creates a control model $\mathcal{M}^{(\varnothing)}$ and a treatment model $\mathcal{M}^{(X)}$ by continued pre-training from the same checkpoint, with a set of sequences to memorize X injected into $\mathcal{M}^{(X)}$'s training data. Our framework explicitly creates a counterfactual state that allows us to study, via causal interventions, what the model would have been if it had not seen a particular sequence.

One hypothesis is that *there are specialized model weights or mechanisms dedicated to recalling the verbatim memorized texts* (Nasr et al., 2023; Chang et al., 2024b; Stoehr et al., 2024). Under this view, preventing verbatim memorization should be straightforward. For example, localizing and intervening on these dedicated components (e.g., a few neurons; Chang et al. 2024b; Maini et al. 2023 or a particular attention head; Stoehr et al. 2024) should remove verbatim memorized texts while preserving model quality. However, recent work indicates that removing verbatim memorized information is challenging. Preventing verbatim memorization during decoding does not stop variants of the memorized texts from being generated (Ippolito et al., 2023), and memorized texts can be retrieved in contexts different from the ones seen in training (Karamolegkou et al., 2023; Ippolito et al., 2023; Kassem et al., 2024). In addition, unlearning via fine-tuning and pruning degrades model quality

[*] Equal advising.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 10711–10732
November 12-16, 2024 ©2024 Association for Computational Linguistics

(Stoehr et al., 2024; Chang et al., 2024b; Maini et al., 2024; Lynch et al., 2024; Chen and Yang, 2023). These findings suggest an alternative view: rather than having specialized weights or mechanisms dedicated to verbatim memorization, *models might be reconstructing memorized sequences using features learned from general language modeling*. This would explain why we are unable to localize memorized texts and why mitigating memorization can fundamentally alter model behaviors.

In this paper, we seek to answer these questions. We develop a framework for studying memorization in controlled settings: given an LM checkpoint $\mathcal{M}$, we continue pre-training $\mathcal{M}$ on the original training data but with specific novel sequences inserted at controlled frequencies. This framework complements existing observational methods, and allows us to decouple factors that potentially affect memorization, including model size, data frequency, and model quality. We use this framework in experiments with the Pythia family of models (Biderman et al., 2023b). Our core findings are as follows: (1) Sequences need to be repeated a non-trivial number of times to be memorized. The perception that a model verbatim memorizes a sequence that occurs once in pre-training is likely an illusion. Later (and presumably better) checkpoints are more likely to verbatim memorize sequences, and even out-of-domain sequences are memorized at non-trivial rates by the best models. (3) Only some tokens in verbatim memorized sequences causally depend on a set of distributed triggering states that encode high-level semantic features, with the rest produced by regular LM decoding. Based on these findings, we develop stress tests to evaluate unlearning methods and find they often fail to remove verbatim memorized information while also degrading model quality.

Overall, these results challenge the view that verbatim memorization stems from specific model weights or mechanisms. Instead, they suggest that verbatim memorization is the result of many interacting factors related to data and language modeling. Thus, removing verbatim memorized information without degrading model quality will be very difficult, especially for our best models.

2 Related Work

Verbatim memorization LLMs can generate long sequences that are exact matches of their training data (Carlini et al., 2021, 2023). Data repetitions, model size, and context length have been identified as contributing factors, mostly through observational studies (Carlini et al., 2023; Karamolegkou et al., 2023; Chen et al., 2024b; Prashanth et al., 2024). However, the mechanisms behind these factors are still not well understood. Prevention measures like fine-tuning, pruning, or string matching often degrade model quality or fail to cover variations of memorized sequences (Stoehr et al., 2024; Chang et al., 2024b; Ippolito et al., 2023). This has motivated attempts to predict memorization before training (Biderman et al., 2023a). We revisit these findings and show that the challenges in prevention are caused by the entanglement between verbatim memorization and general language modeling.

Memorization and generalization Memorizing individual examples is linked to generalization in deep neural models (Arpit et al., 2017; Zhang et al., 2017). Studies on image classifiers show memorizing noisy labels helps long-tail generalization (Feldman and Zhang, 2020; Feldman, 2020). However, verbatim memorization in LLMs differs from these settings, as texts memorized by LLMs are neither long-tail nor noisy. Recent work shows LLMs can memorize data without overfitting (Tirumala et al., 2022); memorization is a gradual process (Dankers et al., 2023; Dankers and Titov, 2024) and is necessary for accuracy (Brown et al., 2021). We investigate the memorization–generalization relationship from the other direction, namely, how generalizable features learned from language modeling play a role in verbatim memorization.

Interpreting memory structures in Transformers MLP layers have been identified as key–value stores for structured knowledge and n-grams (Geva et al., 2021; Dai et al., 2022; Meng et al., 2022; Geva et al., 2023; Voita et al., 2023; Haviv et al., 2023), while Allen-Zhu and Li (2024) shows that attention heads can also store factual knowledge. Yet, how the Transformer stores *free-form texts* has not been well studied, with mixed results that localize memorized texts to MLP layers (Chang et al., 2024b) or attention heads (Stoehr et al., 2024). In this work, we use causal intervention methods (Pearl, 2001, 2009; Beckers and Halpern, 2019; Vig et al., 2020; Geiger et al., 2021) to provide an approximate answer to the crucial counterfactual question of what would have happened if the model had not seen the memorized string (van den Burg and Williams, 2021; Zhang et al., 2023; Feldman and Zhang, 2020; Lesci et al., 2024).

3 A Framework for Studying Verbatim Memorization

We first introduce a framework to study the effects of language modeling quality on verbatim memorization in a tightly controlled setting. This framework adapts the data injection methods of Jagielski et al. (2023) and Carlini et al. (2019), and aims to create minimally different models with and without specific sequences injected into their training data.

Sequence injection We begin with a model checkpoint $\mathcal{M}_i$. Let O_i be the state of the optimizer at checkpoint i, and let D_i be the final datapoint from the dataset D that $\mathcal{M}_i$ was trained on. Using the state $(\mathcal{M}_i, O_i, D_i)$, we create two models. The **control model** $\mathcal{M}^{(\varnothing)}$ continues training $\mathcal{M}_i$ for s steps using the data $D_{[i:i+s]}$, with O_i as the optimizer. For the **treatment model** $\mathcal{M}^{(X)}$, we minimally alter $D_{[i:i+s]}$ to include a set of sequences X that does not otherwise occur anywhere in D. Each sequence in this set is repeated uniformly every m steps from a random offset, replacing the sequence at that point in D, until training step $i + s$.

The framework allows us to independently control three factors: the language model quality of $\mathcal{M}$, the sequences X to be memorized, and the frequency of the target sequence in the training data. Moreover, it creates approximate counterfactuals that allow us to observe what the model would be like if the model had not seen a particular sequence.

Optimizer state O_i To simulate pre-training, we want an optimizer state that reflects the pre-training process prior to step i. Resetting the optimizer would lead to the first few batches having an unduly large impact on the model loss. To achieve this, we first continue training the model $\mathcal{M}_{i-t}$ from the pre-training checkpoint at $i - t$ over examples correspond to the next t steps, using a freshly initialized optimizer. We then use the optimizer state of $\mathcal{M}_{i-t}$ as the optimizer state O_i.

Measuring verbatim memorization We adopt the kl-extractable definition (Carlini et al., 2023) and define the *verbatim memorization length* of an injected sequence $\mathbf{x}$ as the number of tokens in the longest memorized substring. We prompt the model with all substrings in $\mathbf{x}$ of $k = 8, 16, 32, 64$ tokens, where the first 8 tokens of the continuation in $\mathbf{x}$ is **not** a substring in the prompt. For each prompt, we greedy decode the next 64 tokens as the prediction. Among all the predictions, we compute the longest prefix match between the prediction and the actual continuation in $\mathbf{x}$ as the verbatim memorization length l.

4 Experiments

We now report on a sequence of experiments aimed at helping to characterize the nature of verbatim memorization via the following four analyses.[1]

4.1 General Experimental Setup

Models We use checkpoints from the Pythia `160m`, `2.8b`, and `6.9b` models (Biderman et al., 2023b) trained on the Pile (Gao et al., 2020) deduped data.

Injection sequences We curate a set of 100 sequences, each with 256 tokens, sampled from internet content published after the Pile cutoff date. We verify that the overlap between each sequence and the Pile is less than 50 characters (see Appendix B.2). Additionally, we create a set of 100 shuffled sequences by randomly shuffling tokens in each original sequence. The shuffled set preserves the overall vocabulary distribution but with little or no predictable structure.

Realistic injection frequencies To determine realistic frequencies, we study the frequency range that triggers memorization using 5K sequence samples. A sequence is considered memorized if it has a verbatim memorization length of 32 given a prefix of at most 32 tokens. We then hand-select a frequency where the `160m` model produces a mix of the memorized and non-memorized sequences, which is about every 10K to 100K examples. We detail the sampling and counting procedure in Appendix A. Additionally, we observe that at the 6.9B scale, 94% of memorized sequences occur at least 1 in 5M examples, which raises the question of whether a model could memorize a sequence it has seen only once. We address this question in §4.2, finding that purported instances are likely illusory.

Optimizer state In §4.2, we use a freshly initialized AdamW optimizer (Loshchilov and Hutter, 2019). In §4.3 and §4.4, we initialize the optimizer by pre-training on 1M examples.

Additional setup details are in Appendix B.

4.2 The Illusion of Single-shot Verbatim Memorization

Do LLMs verbatim memorize a sequence in pre-training after only seeing the sequence once?

[1] Data and code available at https://github.com/explanare/verbatim-memorization

Type	Percentage	Description
Template	54	Templated texts, where the variable content is usually provided in the prompt
Variation	21	Spacing, punctuation, and textual variants of texts
Induction	17	Texts with inductive patterns, e.g., ordered number sequences or repeating sequences
Composition	8	Texts composed of frequent patterns, with composition rules specified by the prompt

Table 1: Four types of sequences that create the illusion of single-shot verbatim memorization.

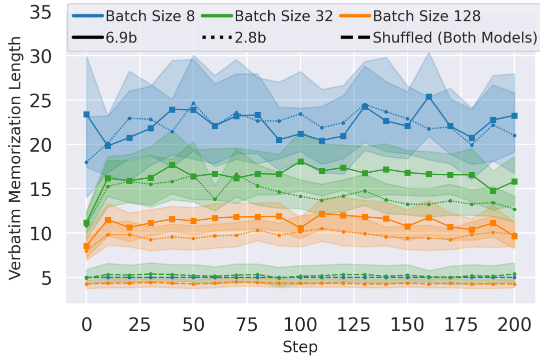


Figure 2: Single-shot verbatim memorization length of the 2.8b and 6.9b models after 200 training steps.

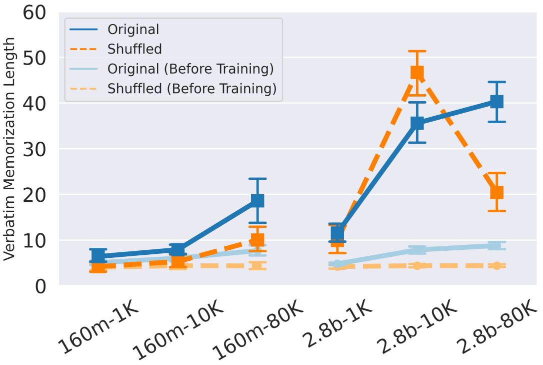


Figure 3: Pythia checkpoint vs. verbatim memorization length of the original and shuffled sequences.

We first manually annotate the 6% low-frequency sequences verbatim memorized by the 6.9b model in §4.1 and identify four patterns that create the illusion of single-shot verbatim memorization in Table 1, with examples shown in Appendix A.2. These seemingly low-frequency sequences are either under-counted due to limitations of string-based matching or simply not verbatim memorized, i.e., a checkpoint produced before the sequence occurs can already generate the sequence verbatim. Prashanth et al. (2024) has also identified similar sequences as "Reconstruction" and "Recollection". These patterns suggest not all tokens in the verbatim memorized sequences are actually memorized; some might be completed by the LM.

One may argue that a memorized sequence that only occurs once in the training data is inherently hard to discover. To complement counting, we directly measure a model's ability to verbatim memorize a sequence after one occurrence.

Setup We train the 2.8b and 6.9b 80K checkpoints for 200 steps, where a sequence to memorize is injected into *the first batch*. We measure the verbatim memorization length every 10 steps.

Results Results are shown in Figure 2, averaged over 16 injection sequences and their shuffled versions. The verbatim memorization length decreases significantly as the batch size increases. Moreover, the verbatim memorization length peaks around 25–100 steps after seeing the injected sequence, likely due to momentum terms in the optimizer (Chang et al., 2024a). Even at the peak, the 6.9b model only verbatim memorizes 12±3.7 tokens from the original sequences at batch size 128. Shuffled sequences are memorized 5±1 tokens regardless of batch size or model size. With a batch size of 1024 in pre-training, it is extremely unlikely that models with a size smaller than 6.9B can just verbatim memorize an arbitrary sequence in a single-shot.

4.3 Better LMs Memorize Longer Sequences

Are better LMs more likely to memorize sequences? Intuitively, better LMs are those that achieve lower perplexity on novel sequences drawn from their training distribution. From this perspective, we might expect them to be better at memorizing such sequences as well, since they simply require fewer bytes on average to encode such sequences (Deletang et al., 2024).

Setup To decouple model quality from model size, we experiment with three checkpoints at 1K, 10K, and 80K steps from the 160m and 2.8b models. We use two injection frequencies for both models: every 50K and 10K examples. For each model run, we pack a set of 4–10 sequences, so that the total number of injected sequences is less than 0.04% of the training data. We measure the verbatim memorization after 40 and 20 occurrences for the two models respectively, as with 20 occurrences the 2.8b model can already memorize longer sequences than the 160m one.

Results Figure 3 (solid blue lines) shows the results for 1 in 50K frequency; results for 10K frequency are in Appendix C.1. The findings are clear: later checkpoints memorize more, and the larger model is able to memorize more, even when seeing the sequences fewer times. Overall, checkpoints corresponding to higher quality models are more likely to memorize the injected sequences.

4.4 Sequences without Structure Are Harder to Memorize

The previous section shows that better models are more capable of memorizing sequences in their training distributions. What about sequences from a different distribution? One hypothesis is that out-of-domain sequences are more likely to be memorized, as they contain rare sequences of tokens that can be used as identifiers to recall the memorized content (Tirumala et al., 2022). The other hypothesis is that in-domain sequences are more likely to be memorized, because they have lower perplexity before training, as in the single-shot case in §4.2.

Setup To investigate which hypothesis holds, we use the set of shuffled sequences, which naturally have a higher perplexity than the original sequences when measured using the model M, i.e., before training on the sequences. We follow the training and evaluation protocol from §4.3.

Results Results are shown in Figure 3 (dashed orange lines). On average, the original sequences drawn from the training distribution are more likely to be verbatim memorized than shuffled sequences, except for the `2.8b-10K` checkpoint. Even though the shuffled sequences are not memorized as well, we do still see the trend in which later checkpoints memorize longer sequences. In terms of perplexity changes, the perplexity of memorized shuffled sequences does decrease faster during training than the perplexity of original sequences.

These findings suggest that the verbatim memorization observed in pre-trained LLMs is more complex than recalling a sequence based on some unique identifiers, as otherwise we would see the models memorize more shuffled sequences. Multiple factors might contribute to the process: general mechanisms to efficiently store free-form texts, as well as structures that favor in-domain sequences. The former might be learned from modeling random sequences such as URLs, UUID, or even digits of π, while the later might emerge for modeling structures in natural languages. We investigate these mechanisms in the following sections.

4.5 Memorization is Triggered by Abstract Model States Distributed Across Tokens

A core question for verbatim memorization is how models encode memorized sequences. We consider two aspects of the question: (1) Which tokens encode the information of the verbatim memorized sequences? (2) Do models encode token-level information (low-level representations) or more abstract states of the model (high-level representations)?

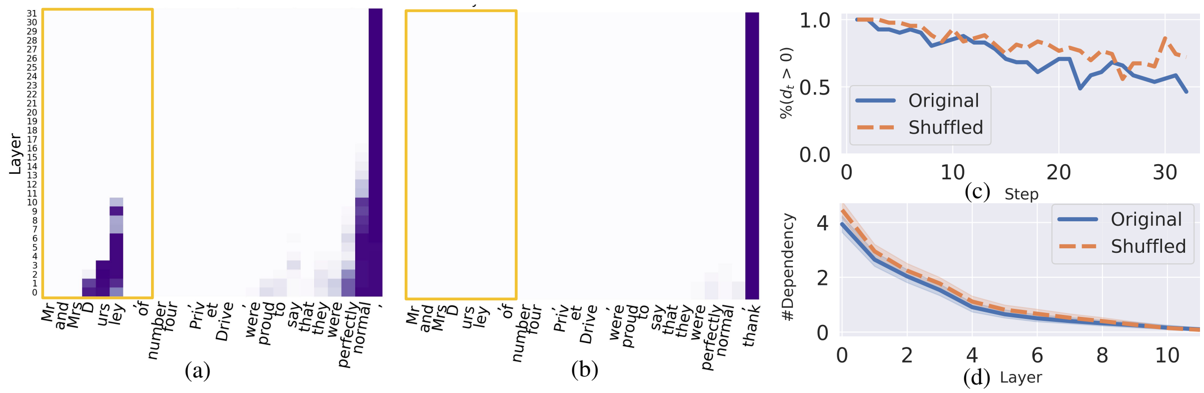
To answer these questions, we seek to identify the causal connections between the sequence that triggers memorization and the tokens in the verbatim memorized sequence. In more detail, consider a treatment model $\mathcal{M}^{(X)}$ that takes as input a *trigger* prefix $\boldsymbol{p} = x_1, \ldots x_n$ and outputs a verbatim memorized sequence $\boldsymbol{s} = x_{n+1}, \ldots x_{n+k}$. From this it follows that the trigger prefix $\boldsymbol{p}$ creates an internal state $\mathcal{S}$ in $\mathcal{M}^{(X)}$ that is sufficient for generating $\boldsymbol{s}$. If verbatim memorized information is *localized* to the trigger $\boldsymbol{p}$, then *every* token in $\boldsymbol{s}$ should have a causal connection to the internal state $\mathcal{S}$.

Interventions To test whether every token in $\boldsymbol{s}$ in fact depends on $\mathcal{S}$, we use *interchange interventions* (also known as activation patching; Geiger et al. 2021; Vig et al. 2020), which is calculated as follows. First, let $\texttt{GetVal}(M(x), l)$ be the representation $\mathbf{v}$ that model M computes at location l when it processes input x. Second, let $M_{l \leftarrow \mathbf{w}}(x)$ be the model that is just like $M(x)$ except that the representations computed at location l have been replaced by the values $\mathbf{w}$. An interchange intervention is one in which the value used in this intervention is one created when the model processes a different input x'. This results in a nesting of $\texttt{GetVal}$ inside the intervention: $M_{l \leftarrow \texttt{GetVal}(M(x'), l)}(x)$. In other words, the interchange intervention replaces the values computed at l with those obtained from processing a different example.

Our interchange intervention focuses on the decoding process given a trigger prefix. Suppose $\mathcal{M}^{(X)}$ has run a forward pass on $\boldsymbol{x} = x_1, \ldots, x_{n+1}, \ldots, x_{n+t-1}$, i.e., the trigger $\boldsymbol{p}$ and the subsequent $t-1$ tokens, and so is going to predict token x_{n+t}. Our interchange intervention replaces the residual stream representations in layer k of a token x_j (for $j \leq n$) in $\boldsymbol{p}$ with residual stream representations extracted at the same layer and token position where the model input is a random sequence $\boldsymbol{r}$ sampled from the Pile:

$$\mathcal{M}^{(X)}_{(j,k) \leftarrow \mathbf{v}}(\boldsymbol{x}) \quad \text{where } \mathbf{v} = \texttt{GetVal}(\mathcal{M}^{(X)}(\boldsymbol{r}), (j,k)) \quad (1)$$

If the next token prediction is causally dependent on the trigger representation, we expect the predicted token to change after this intervention, since



(a) (b) (c) (d)

Figure 4: Causal dependencies between the trigger and verbatim memorized tokens. (a) An example of a memorized token that depends on the trigger (the yellow box). A darker color indicates that the output has a stronger causal dependency on the residual stream at the location. (b) An example of a memorized token that does not depend on the same trigger. (c) The percentage of memorized tokens that causally depend on the trigger decreases by step. (d) For memorized tokens that depend on the trigger, there is on average one causal dependency even at the middle layers.

the chance of a random sampled token having a similar representation as the intervened token is extremely low. Alternatively, if the representation has no causal effect, we expect the output to be the same. Interventions on model representations allow us to measure which tokens have causal effects (our question 1 above), and at which layers the token information was used by the model to decode the next memorized token (our question 2).

Metrics Let l be an intervention location, i.e., a residual stream at the token position i and layer ℓ, and L be the total number of layers. Let p_l represent the percentage of interventions that lead the model to output the verbatim memorized token when intervened on at l. We estimate the causal effect of the trigger on the verbatim memorized token x_{n+t} as a causal dependency score:

$$d_t = 1 - \max_{l \in \{(i,\ell) | 1 \leq i \leq n, 1 \leq \ell \leq L\}} \{p_l\} \qquad (2)$$

The score is between 0 and 1, where 1 means strong causal dependency on the trigger and 0 means no causal dependency. By definition, $d_1 = 1$, since the last layer residual stream at the last token always has causal effects on the first predicted token.

We define the number of dependencies a memorized token x_{n+t} has in a given layer ℓ as $N_{t,\ell}$:

$$N_{t,\ell} = \sum_{l \in \{(i,\ell) | 1 \leq i \leq n\}} \mathbb{1}[p_l > T] \qquad (3)$$

where T is a threshold that we set to 0.1 to filter dependencies with weak causal effects.

Setup We analyze the models in §4.3 trained from the `160m-80K` checkpoint with an injection frequency of 1 in 10M examples. We sample 50 injected sequences (original and shuffled) and compute p_l over 100 random sequences from the Pile.

Results Figure 4 summarizes our results: (1) Not all tokens in the verbatim memorized sequence are causally dependent on the trigger representations, e.g., Figure 4b measured by $1 - p_l$ and Figure 4c measured by d_t. Instead, these tokens often exhibit dependencies that resemble syntactic structures, e.g., the direct object depends on the preceding verb and the closing parenthesis depend on the opening parenthesis. For sequences with no clear structure, memorized tokens depend on more trigger tokens that are relatively rare, a pattern observed in previous work (Tirumala et al., 2022; Stoehr et al., 2024). (2) Most memorized tokens depend on higher-level representations produced by middle layers. In Figure 4d, at layer 4, there still exists on average one dependency. We observe similar patterns in the `6.9b` model (see Appendix C.2).

Overall, these results show that information about the verbatim memorized sequence is (1) distributed across tokens and (2) encoded in abstract states as opposed to token-level features. There is simply *no* representation of the trigger $\boldsymbol{p}$ that causally encodes the *entire* memorized sequence.

Moreover, the fact that not all verbatim memorized tokens are causally dependent on the trigger suggests models might only memorize information about a subset of tokens, filling in the gaps with general language modeling. The verbatim memorized sequence might be *reconstructed* token-by-token, where each token is predicted using different mechanisms depending on the structures involved. This might explain why in-domain sequences are more likely to be memorized. In fact, the two mechanisms we observed – attending to syntactic structures and rare tokens – are identified in Transformers that have not seen or memorized a particular sequence (Tian et al., 2023; Chen et al., 2024a). Lastly, models encode abstract states as opposed to token-level information, which might explain why memorized sequences can be triggered in contexts that are different from those seen in training. We further test this hypothesis in §4.6.

4.6 Verbatim Memorization Leverages General Language Modeling Capabilities

The results of §4.3 and §4.4 provide behavioral evidence that memorization depends on general language capabilities. In this section, we extend the intervention-based methods of §4.5 in an effort to characterize this relationship in terms of the underlying computation they share. The core analytic technique is an interchange intervention that seeks to get a control model $\mathcal{M}^{(\varnothing)}$ to produce memorized strings by intervening with internal states from a minimally different treatment model $\mathcal{M}^{(X)}$.

Our core finding is that, while such interventions do not lead $\mathcal{M}^{(\varnothing)}$ to produce entire memorized sequences, we can get it to produce the first few tokens of such sequences. Moreover, among the interventions at a layer that do produce memorized tokens, more than 50% can still produce the same memorized tokens using model components at the corresponding layer from $\mathcal{M}^{(\varnothing)}$, which are weights learned only from general language modeling.

Cross-model interchange interventions We propose a novel intervention that replaces representations in $\mathcal{M}^{(\varnothing)}$ with corresponding ones in $\mathcal{M}^{(X)}$:

$$\mathcal{M}^{(\varnothing)}_{l\leftarrow\mathbf{v}}(\boldsymbol{p}) \text{ where } \mathbf{v} = \texttt{GetVal}(\mathcal{M}^{(X)}, \boldsymbol{p}, l) \quad (4)$$

Suppose $\boldsymbol{p}$ is a trigger for memorized sequence $\boldsymbol{s}$. If this intervention leads $\mathcal{M}^{(\varnothing)}_{l\leftarrow\mathbf{v}}(\boldsymbol{p})$ to generate parts of $\boldsymbol{s}$, then we have evidence that the memorization behavior was guided in part by the representation at l and in part by the general structure of $\mathcal{M}^{(\varnothing)}$.

It may seem surprising to transfer representations between two models. However, the models begin from the same checkpoint and are trained on almost identical sequences. This weakly suggests that their representations will be compatible. In addition, prior work has shown that even representations from different families of models are interchangeable with some affine transformations (Csiszárik et al., 2021; Ghandeharioun et al., 2024). We also experimentally verify the coherence of these interventions in our results section below.

As in §4.5, we explore intervention sites across all layers, since we do not know a priori where the relevant information might be stored. For each layer, we target both attention and MLP components, which have been identified as related to memorization behaviors in Transformer-based LMs (Geva et al., 2021; Dai et al., 2022; Geva et al., 2023; Stoehr et al., 2024; Allen-Zhu and Li, 2024). We aim to understand to what extent these components reuse computations learned from general language modeling.

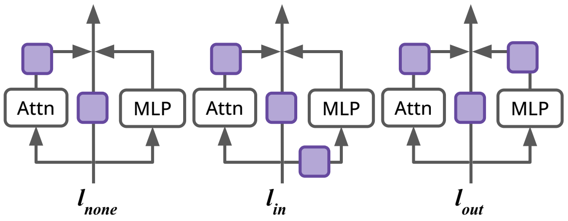


Figure 5: Three sets of cross-model interchange interventions that allow us to measure to what extent models reuse components learned from general language modeling in verbatim memorization.

Metrics For an intervention at location l, let $p_{l,n}$ be the percentage of examples where the first n tokens predicted by $\mathcal{M}^{(\varnothing)}$ match the verbatim memorized tokens generated by $\mathcal{M}^{(X)}$. We consider small values of $n \in [1, 2, 4]$; as we will see, by $n = 4$, success rates have gone to effectively 0.

For each layer, we want to measure whether verbatim memorization reuses computations learned from general modeling, i.e, computations defined by components in $\mathcal{M}^{(\varnothing)}$. We compute $p_{l,n}$ at three sets of intervention locations across all trigger tokens. We use MLP as an example in Figure 5.

- $l_{none,i}$: Attention output at layer i + Residuals at layer $i-1$
- $l_{in,i}$: $l_{none,i}$ + MLP **input** at layer i
- $l_{out,i}$: $l_{none,i}$ + MLP **output** at layer i (i.e., Residuals at layer i)

In $l_{none,i}$, the residual from the treatment model $\mathcal{M}^{(X)}$ is not propagated into the MLP layer of the control model $\mathcal{M}^{(\varnothing)}$. The MLP output is still computed using the MLP input from $\mathcal{M}^{(\varnothing)}$. The locations for attention can be defined symmetrically.

Let $R_{i,n}$ be the percentage of interventions that lead to $\mathcal{M}^{(\varnothing)}$ producing a memorized short prefix of n tokens using only MLP input from the treatment model $\mathcal{M}^{(X)}$, but not the MLP layer weights from the treatment model $\mathcal{M}^{(X)}$:

$$R_{i,n} = \frac{p_{l_{in,i},n} - p_{l_{none,i},n}}{p_{l_{out,i},n} - p_{l_{none,i},n}} \quad (5)$$

$R_{i,n}$ is only meaningful when the denominator is

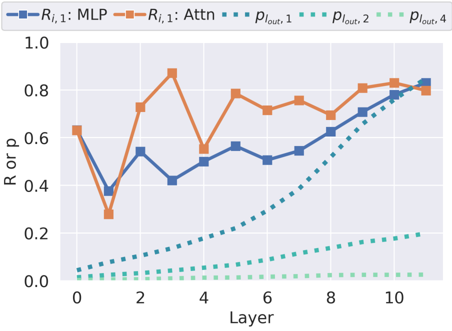


Figure 6: Results of cross-model interventions. Dotted lines: There are interventions that can control $\mathcal{M}^{(\varnothing)}$ to produce the next 1–2 memorized tokens, but not any longer. Solid lines: Among interventions that produce the next memorized token, more than 50% can still produce the same token using components of $\mathcal{M}^{(\varnothing)}$.

sufficiently large, i.e., the layer has causal effects on the next n verbatim memorized tokes. A higher $R_{i,n}$ value suggests that the MLP (or attention) component in $\mathcal{M}^{(\varnothing)}$ plays a similar causal role on the next n memorized tokens as the corresponding component in $\mathcal{M}^{(X)}$. In other words, a sign of leveraging general language modeling capabilities.

Setup We use the `160m` models in §4.3 trained from the step 80K checkpoint, with treatment model data injected at a frequency of every 10K examples. We analyze 2,000 tokens predicted as part of 120 verbatim memorized sequences (including shuffled sequences), which covers about 1000 distinct tokens. About 25% of these verbatim memorized tokens can be correctly predicted by the control model as well. We exclude these from further analysis. However, these tokens suggest that a quarter of the verbatim memorized tokens result from general language modeling. For the remaining tokens, we compute $p_{l_{out,i},n}$ and $R_{i,n}$.

Results Results are shown in Figure 6. We first look at the dotted lines: (1) When intervening on the last layer residuals, $p_{l_{out,11},n} = 85\%$, which validates our intervention setup – representations from the two models are indeed interchangeable for the majority of the inputs. (2) As n increases, $p_{l_{out,i}}$ drops to almost zero, which means interventions on individual model components have little to no causal effect on producing the memorized prefixes. This aligns with our findings in §4.5: Memorized information is distributed across tokens.

For the solid lines, which are $R_{i,1}$ (a setting where a significant percentage of interventions can produce memorized tokens), we find that (1) the majority of attention and MLP layers have $R_{i,1}$ values above 50%, suggesting the $\mathcal{M}^{(X)}$ model is performing similar computations as $\mathcal{M}^{(\varnothing)}$, which are computations learned from general language modeling. In fact, verbatim memorization can still happen with frozen attention heads, i.e., only using attention patterns learned from general language modeling (Appendix C.3). (2) There are a few layers where $R_{i,1}$ is around 30%, i.e., MLP components in layer 1 and 3 and attention in layer 1 and 4, suggesting these components are largely different between $\mathcal{M}^{(\varnothing)}$ and $\mathcal{M}^{(X)}$ and likely store memorized information. Indeed, previous work that uses gradient-based approaches also indicates that lower layers play an important role in verbatim memorization (Stoehr et al., 2024). However, an $R_{i,1}$ around 30% means it is still challenging to fully isolate the memorized information, even just for predicting a single token.

Analysis The ability to leverage computations learned from general language modeling provides an explanation of why higher quality models verbatim memorize more sequences. This also suggests that verbatim memorization is fundamentally intertwined with language modeling capabilities, as the control and treatment models largely share both attention and MLP structures across multiple layers.

5 Stress Testing on Unlearning Verbatim Memorized Texts

Given the nature of the verbatim memorization discussed in §4.5 and §4.6, we propose a suite of automated stress tests to evaluate whether unlearning methods truly remove verbatim memorized information without systematically altering the LM.

5.1 A Stress Testing Dataset

Our stress tests are built on two observations:

- Memorized information is distributed across tokens, hence evaluation should include prompts that cover different spans of a memorized sequence ("Position Perturbations").
- Verbatim memorization is triggered by abstract model states, hence evaluations should cover semantically similar variations of the prompt trained on ("Semantic Perturbations").

Consider a trigger prompt of n tokens $x_1 \dots x_n$ with memorized continuation $x_{n+1} \dots x_{n+k}$ in the original training set (which is also the evaluation set in the unlearning setup). For "Position Perturbations", we generate two sets of perturbed prompts:

$$\{x_1 \dots x_{n+i} \,|\, i \in [0, t]\} \cup \{x_{n-i} \dots x_n \,|\, i \in [t, n)\}$$

Method	Original	Position	Semantic
Gradient Ascent	19±18	35±15	31±21
Sparse Fine-tune	23±20	36±16	34±21
Neuron Pruning	4±4	14±8	11±10

Table 2: The exact match length of model outputs with the original and stress testing prompts. On average, stress testing prompts can extract 10–15 more tokens.

For "Semantic Perturbations", we replace each word or a consecutive sequence of digits (or characters) in the prompt with a similar word.

$$\{x_1 \dots s_i \dots x_n \mid i \in [1, n], s_i \in \mathcal{S}(x_i)\}$$

where $\mathcal{S}(x_i)$ is a set of words similar to x_i. Example stress tests are in Appendix C.4.

5.2 Evaluation

We evaluate the gradient ascent, sparse fine-tuning, and pruning methods of Stoehr et al. (2024) and Chang et al. (2024b). These methods have been shown to prevent models from generating verbatim memorized texts on the fine-tuned prompts, at the cost of increasing perplexity on other texts (Chang et al., 2024b; Stoehr et al., 2024).

Setup We follow the setup in Stoehr et al. (2024) (see Appendix B.6). Given a 50-token trigger prompt and a 50-token continuation memorized by the GPT-Neo 125M model, the goal is to unlearn the continuation while retaining model quality on other prompts. For each sequence, we generate ≈1K perturbed prompts with $t = 20$ for Position Perturbations and use ChatGPT to generate around 10 similar word substitutions per word for Semantic Perturbations. For both original and stress test prompts, we report the longest continuation that matches the memorized sequence. For stress test prompts, the length is max-pooled over all prompts.

Results Table 2 shows the results, with full length distributions shown in Appendix C.4. On average, the perturbed prompts increase the exact match length by 10–15 tokens. For gradient ascent and sparse fine-tuning, the stress tests increase the fully extractable sequences (i.e., exact match of 50 tokens) from 22% to 56%. The neuron pruning method is more robust to the stress tests. However, it often leads to degeneration on the perturbed prefixes, e.g., outputting repetitive texts. Overall, while these unlearning methods largely prevent models from generating the verbatim memorized sequence given a particular prefix, they do not completely remove the verbatim memorized information – the model can still generate the memorized texts when prompted with variants of the prefix.

6 Discussion and Conclusion

Verbatim memorization is a pressing issue for LM research, as it has ramifications for privacy, copyright, and other legal issues. Thus, one might hope that we will find ways to identify and control memorization. The present paper suggests that such control may be extremely difficult to achieve because verbatim memorization is thoroughly intertwined with general language modeling quality. For example, given current training procedures, LMs will memorize more strings as their quality improves. Strings that resemble those from the LM's training data are more likely to be memorized (§4.3), but even OOD strings (which may include private identifiers, usual metadata patterns, etc.) are memorized at non-trivial rates by our best models (§4.4).

In light of these findings, one might simply accept that LMs will memorize strings and try to mitigate memorization by blocking specific triggering strings. Unfortunately, this method is bound to have very low recall. As we showed in §4.5, the notion of a trigger is extremely complex. Overall, the trigger is actually a set of distributed model-internal states that encode generalizable high-level features that numerous inputs can lead to. In §4.6, we deepened this result by showing that even a control model that has never seen a specific memorized input **x** can be made to produce parts of **x** via an intervention from a model that has memorized **x**. In §5, we show the practical implications of these distributed, abstract triggering states on unlearning methods, which lead to failures in removing verbatim memorized information or degrading general model quality. These results all point to the idea that generating memorized strings is in part simply language model decoding as usual.

More broadly, these findings suggest that "verbatim memorization" is something of a misnomer, as the phenomenon involves memorization of more abstract model states as opposed to only memorization of token-level information. Thus, to be successful, future attempts to control memorization will likely require new techniques for characterizing and controlling these abstract model states. Such techniques are likely to greatly improve our understanding of LLMs in general.

Limitations

Our work contributes to understanding verbatim memorization behaviors in LLMs, an important problem that has practical implications and applies to almost all LLMs trained on large-scale web corpora. However, constrained by the availability of fully open sourced LLMs (i.e., LLMs with training dataset, checkpoints, and training hyperparameters fully available), we only conducted experiments on the Pythia family of models, focusing on model sizes up to `2.8b`. As more fully open source models come out, such as OLMo,[2] we would like to see if our findings on Pythia models generalize to other model families.

One important finding of our paper is that verbatim memorization actually involves memorization of abstract model states as opposed to just token-level information. This raises the concern of whether focusing on verbatim memorization reveals the full scale of what models actually memorize. LLMs could memorize long sequences of abstract states as well, which might remain undetected if we only focus on verbatim memorization. For example, models memorize syntactic templates (Shaib et al., 2024). We discuss these findings in §6.

For verbatim memorization treatments, our discussion is focused on post-training treatments, including unlearning (§5) and string-based matching (§6). If we consider the broader LLM development cycle, there are alternative approaches to the verbatim memorization problem, such as deduplication of training data (Lee et al., 2022), interventions during training, e.g., modifying the loss function (Hans et al., 2024), or even building an ecosystem that properly attributes the value of training data to its creators. We hope the findings from our work will help motivate this community to explore more solutions in these spaces.

Acknowledgments

The authors gratefully acknowledge the contributors of fully open LLMs and the developers of indexing tools for LLMs training corpora. This work would not be possible with fully open LLMs like the Pythia suite and dataset indexing tools like the Data portraits[3] and infini-gram.[4]

We would also like to thank Atticus Geiger, Jiaao Chen, Robin Jia, John Hewitt, Ken Liu, Zhengxuan Wu, and Aryaman Arora for insightful discussion and feedback. This work is supported in part by a grant from ONR and the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

[2] https://allenai.org/olmo

[3] https://dataportraits.org/

[4] https://huggingface.co/spaces/liujch1998/infini-gram

References

Zeyuan Allen-Zhu and Yuanzhi Li. 2024. Physics of language models: Part 3.3, knowledge capacity scaling laws.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. 2017. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR.

Sander Beckers and Joseph Y. Halpern. 2019. Abstracting causal models.

Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Gregory Anthony, Shivanshu Purohit, and Edward Raff. 2023a. Emergent and predictable memorization in large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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. 2023b. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Gavin Brown, Mark Bun, Vitaly Feldman, Adam Smith, and Kunal Talwar. 2021. When is memorization of irrelevant training data necessary for high-accuracy learning? In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 123–132, New York, NY, USA. Association for Computing Machinery.

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, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284, Santa Clara, CA. USENIX Association.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association.

Hoyeon Chang, Jinho Park, Seonghyeon Ye, Sohee Yang, Youngkyung Seo, Du-Seong Chang, and Minjoon Seo. 2024a. How do large language models acquire factual knowledge during pretraining?

Ting-Yun Chang, Jesse Thomason, and Robin Jia. 2024b. Do localization methods actually localize memorized data in LLMs? A tale of two benchmarks.

Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L Leavitt, and Naomi Saphra. 2024a. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in MLMs. In *The Twelfth International Conference on Learning Representations*.

Bowen Chen, Namgi Han, and Yusuke Miyao. 2024b. A multi-perspective analysis of memorization in large language models.

Jiaao Chen and Diyi Yang. 2023. Unlearn what you want to forget: Efficient unlearning for llms. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Tong Chen, Akari Asai, Niloofar Mireshghallah, Sewon Min, James Grimmelmann, Yejin Choi, Hannaneh Hajishirzi, Luke Zettlemoyer, and Pang Wei Koh. 2024c. Copybench: Measuring literal and non-literal reproduction of copyright-protected text in language model generation.

Adrián Csiszárik, Péter Kőrösi-Szabó, Ákos Matszangosz, Gergely Papp, and Dániel Varga. 2021. Similarity and matching of neural network representations. In *Advances in Neural Information Processing Systems*, volume 34, pages 5656–5668. Curran Associates, Inc.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.

Verna Dankers and Ivan Titov. 2024. Generalisation first, memorisation second? memorisation localisation for natural language classification tasks. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 14348–14366, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Verna Dankers, Ivan Titov, and Dieuwke Hupkes. 2023. Memorisation cartography: Mapping out the memorisation-generalisation continuum in neural machine translation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8323–8343, Singapore. Association for Computational Linguistics.

Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. 2024. Language modeling is compression. In *The Twelfth International Conference on Learning Representations*.

Vitaly Feldman. 2020. Does learning require memorization? A short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 954–959, New York, NY, USA. Association for Computing Machinery.

Vitaly Feldman and Chiyuan Zhang. 2020. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems*, volume 33, pages 2881–2891. Curran Associates, Inc.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB dataset of diverse text for language modeling. ArXiv:2101.00027.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. 2021. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 9574–9586.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. 2023. Dissecting recall of factual associations in auto-regressive language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12216–12235, Singapore. Association for Computational Linguistics.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. 2024. Patchscopes: A unifying framework for inspecting hidden representations of language models.

Abhimanyu Hans, Yuxin Wen, Neel Jain, John Kirchenbauer, Hamid Kazemi, Prajwal Singhania, Siddharth Singh, Gowthami Somepalli, Jonas Geiping, Abhinav Bhatele, and Tom Goldstein. 2024. Be like a goldfish, don't memorize! mitigating memorization in generative llms.

Adi Haviv, Ido Cohen, Jacob Gidron, Roei Schuster, Yoav Goldberg, and Mor Geva. 2023. Understanding transformer memorization recall through idioms. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 248–264, Dubrovnik, Croatia. 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.

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*.

Antonia Karamolegkou, Jiaang Li, Li Zhou, and Anders Søgaard. 2023. Copyright violations and large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7403–7412, Singapore. Association for Computational Linguistics.

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.

Jooyoung Lee, Thai Le, Jinghui Chen, and Dongwon Lee. 2023. Do language models plagiarize? In *Proceedings of the ACM Web Conference 2023*. ACM.

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.

Pietro Lesci, Clara Meister, Thomas Hofmann, Andreas Vlachos, and Tiago Pimentel. 2024. Causal estimation of memorisation profiles.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Aengus Lynch, Phillip Guo, Aidan Ewart, Stephen Casper, and Dylan Hadfield-Menell. 2024. Eight methods to evaluate robust unlearning in llms.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C Lipton, and J Zico Kolter. 2024. Tofu: A task of fictitious unlearning for llms. *arXiv preprint arXiv:2401.06121*.

Pratyush Maini, Michael C. Mozer, Hanie Sedghi, Zachary C. Lipton, J. Zico Kolter, and Chiyuan Zhang. 2023. Can neural network memorization be localized? In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Marc Marone and Benjamin Van Durme. 2023. Data portraits: Recording foundation model training data. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

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.

Judea Pearl. 2001. Direct and indirect effects. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 411–420, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Judea Pearl. 2009. *Causality*. Cambridge University Press.

USVSN Sai Prashanth, Alvin Deng, Kyle O'Brien, Jyothir SV, Mohammad Aflah Khan, Jaydeep Borkar, Christopher A. Choquette-Choo, Jacob Ray Fuehne, Stella Biderman, Tracy Ke, Katherine Lee, and Naomi Saphra. 2024. Recite, reconstruct, recollect: Memorization in lms as a multifaceted phenomenon.

Chantal Shaib, Yanai Elazar, Junyi Jessy Li, and Byron C. Wallace. 2024. Detection and measurement of syntactic templates in generated text.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, Los Alamitos, CA, USA. IEEE Computer Society.

Niklas Stoehr, Mitchell Gordon, Chiyuan Zhang, and Owen Lewis. 2024. Localizing paragraph memorization in language models.

Yuandong Tian, Yiping Wang, Beidi Chen, and Simon Shaolei Du. 2023. Scan and snap: Understanding training dynamics and token composition in 1-layer transformer. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kushal Tirumala, Aram H. 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*.

Gerrit van den Burg and Chris Williams. 2021. On memorization in probabilistic deep generative models. In *Advances in Neural Information Processing Systems*, volume 34, pages 27916–27928. Curran Associates, Inc.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.

Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2023. Neurons in large language models: Dead, n-gram, positional.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*.

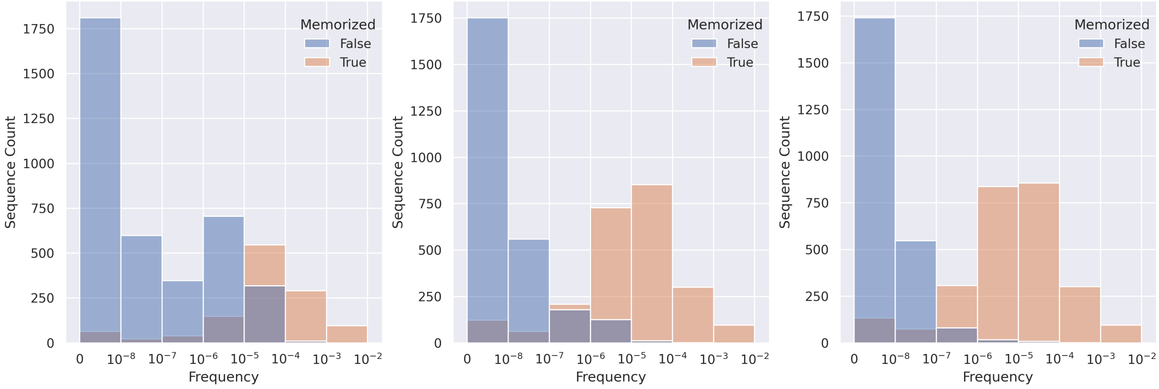
Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2023. Counterfactual memorization in neural language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

A Sequence Frequency vs. Verbatim Memorization

A.1 Choice of Sequence Injection Frequencies

To estimate a realistic frequency for sequence injection, we need to know roughly what percentage of sequences in the Pile are memorized at each frequency range. The deduped version of Pile contains about 98M sequences, each of length 2048 tokens. Ideally, one would build an index of the entire corpus to count substrings, as is done in Carlini et al. (2023); Liu et al. (2024). However, the storage and computation required to build an index is costly. We employ a sampling-based approach instead.

Sampling We first describe how to sample a relatively small set of sequences to estimate memorization rates at each frequency range. We start with random sampling 1M sequences of length 128 from the Pile and compute verbatim memorization length using the `pythia-6.9b-deduped` model. For a sequence to be considered memorized, the sequence must have a verbatim memorization length of at least 32, (i.e., there must exist a substring of length ≤ 32 tokens, such that when prompted with this substring, the model outputs a continuation that matches the next 32 tokens or more). Among the 1M sequences, there are about 9K memorized sequences and 991K non-memorized sequences. Next, we randomly sample 2.5K memorized and 2.5K non-memorized sequences, which means that we downsample the non-memorized sequences 110 times relative to memorized ones. For each sequence, we further sample a substring of 16 tokens. For memorized sequences, the 16 tokens are sampled from the memorized substring, i.e., the model outputs instead of the prompts. We refer to these 5K 16-token sequences as probes.



(a) `pythia-160m-deduped` (b) `pythia-2.8b-deduped` (c) `pythia-6.9b-deduped`

Figure 7: Sequence frequency distribution (on a logarithmic scale) of 2.5K memorized and 2.5K non-memorized sequences, randomly sampled from the 98M Pythia deduped training data. The non-memorized sequences (blue bars) are downsampled 110 times relative to memorized sequences (orange bars).

Counting We uniformly sample about 10M sequences from the Pile deduped dataset to estimate the frequency of each probe. The 10M sequences are sampled at every 10K training step starting at step 0, with a total of 10×1000×1024 sequences of length 2048 tokens. We count the number of occurrences of each probe in the 10M sequences. These probes indeed cover a wide range of frequencies from 0 to 5×10^{-3}. The full distribution is shown in Figure 7.

Evaluating models For each model, we measure the verbatim memorization length on the 5K set of probes. The distribution of memorization length is shown in Figure 7. Aligned with findings from Carlini et al. (2023) and Prashanth et al. (2024), we observe that, as model size increases, the median frequency of memorized sequences decreases from 4×10^{-5}, 1×10^{-5}, to 9×10^{-6}. As we mainly experiment with the `160m` model, we choose two frequencies where there is a mix of memorized and non-memorized sequences: 2×10^{-5} (which is at about the bottom 25th percentile, where sequences are more likely to be non-memorized) and 1×10^{-4} (which is around the top 25th percentile, where sequences are very likely memorized).

```
Template
PROMPT: imGroupExternalMembershipManager getExternal
OUTPUT: MembershipManager() {\n return externalMembershipManager;\n }\n\n public
void setExternalMembershipManager
PROMPT: madesimple.statoil\n\ncmsmadesimple.lips
OUTPUT: y\n\ncmsmadesimple.next\n\ncmsmadesimple.nextdirect\n\ncmsmadesimple.ubs
\n\ncmsmadesimple.war

Variation
PROMPT: the testimony.\n\n Rule 702 which
OUTPUT:  governs  the  admissibility  of  expert  testimony  provides:\n\n\n  If
scientific, technical, or other specialized knowledge will assist the trier of
fact\n to understand the
PROMPT: 13: Thou shalt tread upon the lion and adder: the young
OUTPUT: lion and the dragon shalt thou trample under feet. 14: Because he hath
set his love upon me, therefore will I deliver him: I will set

Induction
PROMPT: 01 3600\n265 3586 3587 3602 3601\n266 35
OUTPUT: 87 3588 3603 3602\n267 3588 3589 3604 3603\n268 3589 3590 3605 3604\n269
3590 3591 3606 3605
PROMPT: ang12.bdf batang12b.bdf \\\n\t\tbatang14.bdf batang14b.bdf batang16.
OUTPUT: bdf batang16b.bdf \\\n\t\tbatang18.bdf batang18b.bdf batang20.bdf batang

Composition
PROMPT: normal; AST: aspartate aminotransferase (i.e. SGOT:
OUTPUT: serum glutamic oxaloacetic transaminase); ALT: alanine aminotransferase
(i.e. SGPT: serum glutamic pyruvic transaminase
PROMPT: G), tenofovir alafenamide/emtricitabine/bic
OUTPUT:   tegravir   (TAF/FTC/BIC),   and   tenofovir   alafenamide/emtricitabine/
rilpivirine (TAF
```

Figure 8: Examples of the single-shot verbatim memorization illusion. Each example is a sequence that occurs once or twice in the `pythia-6.9b-deduped` training data and can be generated by the model verbatim. However, these sequences are likely not learned from a single instance or simply not verbatim memorized – even with a model checkpoint produced before the training step where the memorized sequence occurs, the model can already output the "memorized" sequence or a close variant.

A.2 Examples of the Single-shot Verbatim Memorization Illusion

Figure 8 shows examples of sequences that only occur in the Pile once or twice according to the infini-gram tool[5] and would be considered as verbatim memorized based on the most commonly used extractability definition (Carlini et al., 2021, 2023; Prashanth et al., 2024), i.e., a memorized sequence of 32 tokens can be extracted with a prefix length of 32 tokens. In reality, these sequences are either under-counted due to limitations of string-based matching or simply not verbatim memorized – a checkpoint produced before the step where the sequence occurs can already generate a close variant of the sequence.

These findings suggest that a model generates a sequence that only occurs once in the training data does not necessarily mean that the model verbatim memorized a sequence after one exposure. As shown in §4.5 and §4.6, these sequences may well be "reconstructed" by the general language model.

[5] https://huggingface.co/spaces/liujch1998/infini-gram

B Details of Experiment Setup

B.1 Pre-training Data

We use the Pile deduped version released here,[6] which contains training data in the exact order they were seen by the Pythia deduped models. For our training runs, we use the data from step 80K to 82K, which contain 2M training examples that has not been seen by any of the checkpoints that we experimented with (except for individual examples with duplicates).

B.2 Injection Sequences

Data sources We sampled 100 documents from the Internet that are published after Dec 31th 2020, i.e., the Pile corpus cutoff date. These documents are from five sources that have clear publication timestamps: Wikipedia,[7] BBC News,[8] GitHub,[9] open-access academic papers on ArXiv[10] and Nature,[11] and quotes from novels.[12] All these sources are covered in the original training corpus. For Wikipedia, we sample articles from 2023 categories curated by Wikipedia, for example, the new product category.[13] For BBC news, we use the preprocessed corpus on Huggingface.[14] For GitHub, we use code samples from three new programming languages released after 2020: Mojo,[15] Gleam,[16] and Carbon.[17]

Verify a sequence is not in the Pile In our study, an important criterion for injected sequences is that they do not have significant overlap with the pre-training corpus. This is partially ensured by the document publication date. However, we conduct additional verification.

We use two recently open sourced tools that create a searchable index of the Pile. We primarily rely on Data portraits,[18] which directly checks for overlap between a query text and the Pile corpus using Bloom filters computed from 50-character hashes (Marone and Van Durme, 2023). Bloom filters guarantee no false negatives, however, there will be false positives, i.e., 50-char texts that are not in the Pile but are marked as overlaps. We further confirm these false positives using infini-gram. With both tools, we verify that none of the documents have an overlap with the Pile of more than 50 characters.

B.3 Model Checkpoints

For the sequence injection and the causal dependency experiments, we use the 1K, 10K, 40K, 80K, and the final checkpoints from `pythia-160m-deduped`[19], `pythia-2.8b-deduped`[20], and `pythia-6.9b-deduped` models.[21] For the unlearning stress test evaluation, we follow the setup in Stoehr et al. (2024) and use `gpt-neo-125m`,[22] which is also pre-trained on the Pile.

B.4 Setup for the Single-shot Verbatim Memorization Experiment in §4.2

We randomly sample 16 sequences from the 100 injection sequences curated in Appendix B.2. For each injection sequence, we use the first 224 tokens instead of the full 256 tokens, i.e., a window size of 224, so that we can fit a batch of 32 sequences on a single GPU. In general, with a fixed batch size, a smaller window size makes verbatim memorization more likely to happen, since there are fewer tokens in the batch. Given the actual window size in pre-training is 2048, the verbatim memorization length after a

[6] https://huggingface.co/datasets/EleutherAI/pile-deduped-pythia-preshuffled

[7] https://www.wikipedia.org/

[8] https://www.bbc.com/news

[9] https://github.com/

[10] https://arxiv.org/

[11] https://www.nature.com/

[12] https://www.goodreads.com/

[13] https://en.wikipedia.org/wiki/Category:Products_introduced_in_2023

[14] https://huggingface.co/datasets/RealTimeData/bbc_news_alltime

[15] https://github.com/modularml/mojo

[16] https://github.com/gleam-lang/gleam

[17] https://github.com/carbon-language/carbon-lang

[18] https://pile.dataportraits.org/

[19] https://huggingface.co/EleutherAI/pythia-160m-deduped

[20] https://huggingface.co/EleutherAI/pythia-2.8b-deduped

[21] https://huggingface.co/EleutherAI/pythia-6.9b-deduped

[22] https://huggingface.co/EleutherAI/gpt-neo-125

single-shot is likely even smaller than what we observe in our experiment. We experiment with a batch size of 8, 32, and 128.

We use a freshly initialized AdamW optimizer (Loshchilov and Hutter, 2019), wrapped with the ZeroRedundancyOptimizer,[23] which is used in Pythia training to reduce memory usage (Biderman et al., 2023b). The learning rate is set to a constant value of 1×10^{-4}, with other optimizer parameters, e.g., learning rate decay, beta, set to the default values in PyTorch library.[24] We choose a learning rate that is about twice as large as the actual learning rate at step 80K in pre-training for both the `2.8b` and `6.9b` models, with the consideration that a higher learning rate is more likely to produce a large enough gradient update to memorize the injection sequence in a single step.

B.5 Setup for the Model Quality vs Verbatim Memorization Experiment in §4.3 and §4.4

Ideally, we want to match the exact pre-training setup that Pythia models used. However, we are constrained by the computation resources available to us. Hence, we choose the closest hyperparameters to the ones used in Pythia pre-training that allow us to fit the model training on a single GPU. Our expectation is that the effects of hyperparameters will be minimized as long as the control and the treatment models use the same set of hyperparameters.

Window size We use a window size of 256 tokens for all experiments reported in the paper except the single-shot experiment, a window size that still allows some long range dependencies in the training data. The original window size is 2048.

Batch size We use a batch size of 128 examples for the `160m` models and a batch of size of 40 examples for the `2.8b` models. These batch size are chosen such that we can fit model training on a single GPU. The original batch size used in pre-training is 1024.

Optimizer state As we do continued training from different model checkpoints, we experiment with different initial optimizer states and initial learning rates based on the original Pythia learning rate schedule. To initialize the optimizer state, we pre-train from either step 0K (i.e., 1K step before the earliest checkpoint in our experiment) or step 79K (i.e., 1K step before the latest checkpoint in our experiment) for 1M examples. We observe that these two initial states do not affect which checkpoints verbatim memorize more sequences. Thus, when comparing models trained from two different checkpoints, we use the same optimizer state for both. For learning rate, we use the learning rate at the `80K` checkpoint for each model family, namely 2.79×10^4 for `160m` models and 7.46×10^5 for `2.8b` models. We observe the learning rate affects all checkpoints equally, with larger learning rates leading to more memorization. We keep the learning rate constant throughout the training, as the amount of data we trained on only corresponds to 1–2K steps in the original training process.

B.6 Unlearning Method Hyperparameters

For gradient ascent and sparse fine-tuning, we use the implementation from Stoehr et al. (2024).[25] We follow the hyperparameters here,[26] namely, we run optimization for 10 steps using a learning rate of 1×10^{-5} and a weight decay of 1.0. For sparse fine-tuning, we only fine-tune 0.1% of weights with the highest gradient.

For neuron pruning, we use the implementation from Chang et al. (2024b).[27] We prune 0.1% of the neurons. The L1 penalty is set to 1000. We find that higher L1 penalty leads to degeneration. We run optimization for 1000 steps using a learning rate of 1×10^{-2}. This set of hyperparameters leads to a Δ self-accuracy of -0.248 and Δ neg-accuracy of -0.094 on the 90 sequences to unlearn.

[23] https://pytorch.org/tutorials/recipes/zero_redundancy_optimizer.html

[24] https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html

[25] https://github.com/googleinterns/localizing-paragraph-memorization

[26] https://github.com/googleinterns/localizing-paragraph-memorization/blob/main/notebooks/3%20editing/fine-tuning.ipynb

[27] https://github.com/terarachang/MemPi

B.7 Computation Cost

All models are trained on NVIDIA A100 GPUs. For models in §4.2, the training is distributed across multiple GPUs, with a local batch size of 32. For models in §4.3 and §4.4, the training is on a single GPU. The training of `2.8b` models over 1M examples takes about 16 hours, while the training of `160m` models takes about 3 hours.

C Additional Experiment Results

C.1 Additional Results on Checkpoint vs. Verbatim Memorization Length

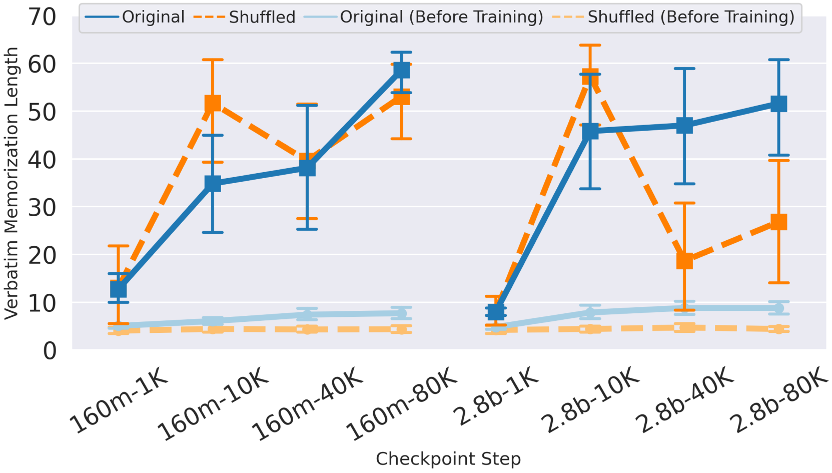


Figure 9: Checkpoint vs. verbatim memorization length of original and shuffled sequences, with a sequence frequency of every 10K examples.

In Figure 9, we show the results of verbatim memorization length when continue pre-training from different checkpoints of the `160m` model and the `2.8b` model with a sequence injection frequency of 1 in 10K examples. This is a frequency that both models are expected to memorize most of the injection sequences. We still see the consistent trend that we observed in §4.3 on in-domain sequences: later checkpoints memorize longer sequences. The gap between shuffled sequences and original sequences is narrowed, especially on the `160m` models, possibly because the model is seeing the injection sequences more frequently. For the `2.8b` models, which see the injection sequences fewer times, shuffled sequences are still harder to memorize than the original ones for all checkpoints except the 10K step.

C.2 Additional Results on Causal Dependencies

Behavioral evidence of verbatim memorization is triggered by abstract model states In Figure 10, we show that when prompted with prefixes sharing similar high-level features, e.g., synonyms or proper nouns belong to the same category, the `6.9b` model can produce the memorized continuation. Semantically similar prefixes do not always trigger verbatim memorization, nor does verbatim memorization strictly require prefixes semantically similar to the one in training, however, semantically relevant substitutions do have a higher probability to trigger the verbatim memorized continuation than random substitutions.

Overall, the trigger is a set of *distributed abstract states* and does not require a particular token to be presented in the prefix, i.e., the verbatim memorization is not triggered by a single n-gram match. This finding motivates the stress tests we developed in §5.

Results of the `6.9b` model In Figure 11, we show the causal dependency results of the pre-trained `pythia-6.9b-deduped` model on 50 memorized sequences sampled from the 5K sequences in §4.1. The results are consistent with what we observed from the `160m` models trained using our sequence injection framework – namely, not all verbatim memorized tokens depend on the trigger sequences. Moreover, for memorized tokens that depend on the trigger, the dependencies are also around middle layers, suggesting high-level features are involved in the verbatim memorization.

The Original Trigger Prefix

Mr and Mrs Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much

Trigger Prefixes with Similar High-level Features

Mrs and Mr Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much

The Dursley family, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Weasley, residing at four Privet Drive, were proud to say they were perfectly normal, thank you very much

Mr and Mrs Slytherin, of number twenty-one, Privet Drive, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Dursley, of #4, Privet Drive, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Dursley, of number ten, Privet Drive, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Dursley, of Privet Drive, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Dursley, of number four, Oak Street, were proud to say that they were perfectly normal, thank you very much

Mr and Mrs Dursley, residing at four Privet Drive, were delighted to assert they were perfectly normal, thank you very much

The Dursley family, of number four, Privet Drive, were pleased to declare that they were perfectly normal, thank you very much

Non-Trigger Prefixes with Similar or Different High-level Features

Mr and Mrs Kingsley, of number four, Privet Drive, were proud to say that they were the proud parents of a bouncing baby boy.

Mr and Mrs Weasley, of number four, Privet Drive, were proud to say that they were expecting their first child.

Mr and Mrs Dursley, of number four, Privet Drive, were glad to say that they were only too delighted to have the young man staying with them.

Figure 10: Examples of trigger prefixes that lead to similar abstract states, i.e., similar high-level semantic features. The gray texts are the prompts. The green texts are the memorized continuations. The red texts are the non-memorized continuations.

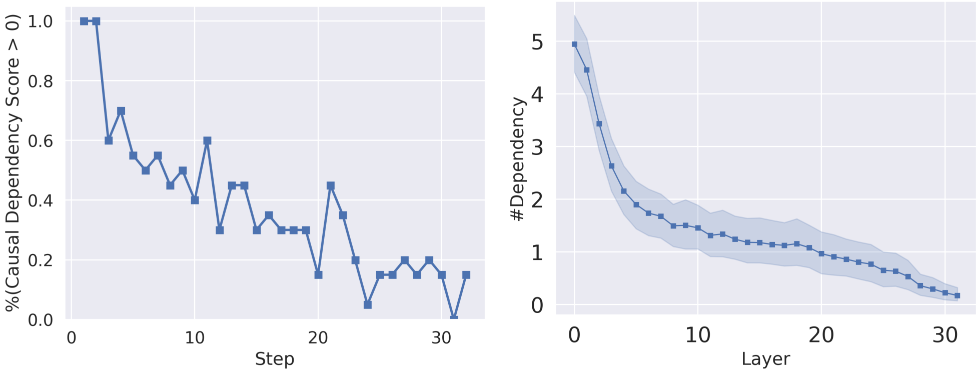


Figure 11: Causal dependencies between memorized tokens and tokens in the trigger for the 6.9b model.

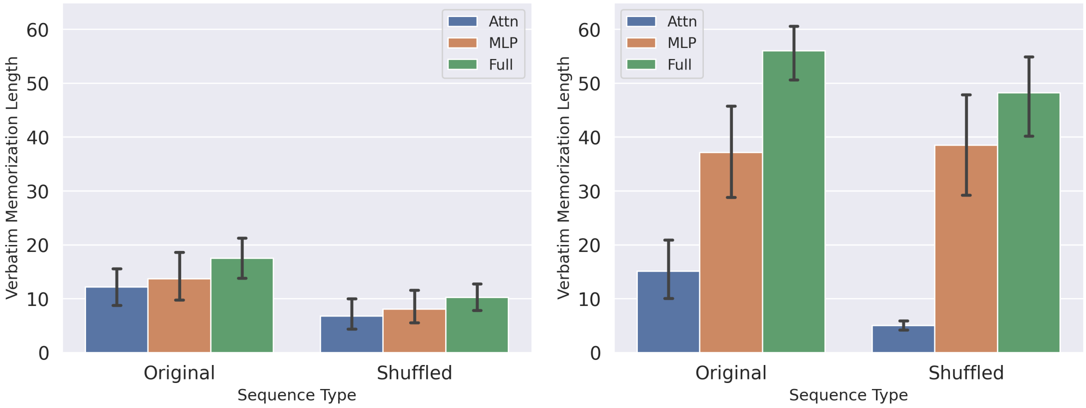


Figure 12: Trainable components vs. verbatim memorization length of original and shuffled sequences. Models are trained from the `160m` model checkpoint at step 80K with two different data injection frequencies (every 50K and 10K examples).

C.3 Verbatim Memorization Can Still Happen with Frozen Attention Heads

Experiments in §4.6 show that both attention and MLP components are involved in verbatim memorization. We now investigate which components are strictly necessary for verbatim memorization, taking the capacity of these components into account.

Setup We conduct an ablation study by training three sets of models using the `160m` model checkpoints: (1) only MLP layer weights are trainable (2) only attention head weights are trainable (3) all weights are trainable. For Pythia models, which use the GPTNeoX architecture, the MLP layers contain 35% of model weights while attention heads contain 17% of model weights. We experiment with two sequence frequency of 1 in 50K and 1 in 10K.

Results Results are shown in Figure 12. With a frequency of every 50K examples, neither MLP-only nor attention-only models can verbatim memorize a sequence. However, at the frequency of every 10K examples, the model with frozen attention heads surprisingly can verbatim memorize sequences of 40 tokens on average, which is about 80% of the tokens memorized by a model with all weights trainable. These results suggest that MLP layers are strictly necessary for verbatim memorization, while attention mechanisms learned from general language modeling can largely be reused assuming the sequence to memorize occurs frequently enough in training.

C.4 Additional Results on Stress Testing Unlearning Methods

In Figure 13, we show examples of the original prompt and the perturbed stress testing prompts, along with the model output before and after unlearning. In Figure 14, we show the verbatim memorization length distribution shift when evaluate with the original prompts and the stress testing prompts.

D License

For artifacts used in this work, the Pythia models are licensed under Apache-2.0 License. The Pile dataset is licensed under MIT License. Our use of the model and the dataset are permitted under the license.

Unlearning with Gradient Ascent

Original Test

PROMPT: 0f86e5b48e01b996cadc001622fb5e363b421”,\n ”uncleHash” : ”0x1dcc4de8

OUTPUT: dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347”\n },\n

UNLEARNED: 67**b567b6ccd41ad312451b9**24af0a0af82a49f0d2c4934\n }\n }\n}\n2:{

Position Perturbations

PROMPT: 5b48e01b996cadc001622fb5e363b421”,\n ”uncleHash” : ”0x1dcc4de8

OUTPUT: dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347”\n },\n

UNLEARNED: **dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347”\n },\n**

Semantic Perturbations

PROMPT: e105b48e01b996cadc001622fb105e363b421”,\n ”uncleHash” : ”0x1dcc4de8

OUTPUT: dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347”\n },\n

UNLEARNED: **dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347”\n },\n**

Unlearning with Gradient Ascent

Original Test

PROMPT: NOT TO BE PUBLISHED IN OFFICIAL REPORTS\n California Rules of Court, rule 8.1115(a), prohibits courts and parties from citing or relying on opinions not certified for\n publication

OUTPUT: or ordered published, except as specified by rule 8.1115(b). This opinion has not been certified for publication\n or ordered published for purposes of rule 8.1115.\n\n\n

UNLEARNED: **or ordered published, except as specified by rule 8.** coli. This Court has not. All opinions are not treated as a whole, and hence opinions are not certified for purposes of\n publication.\n

Position Perturbations

PROMPT: NOT TO BE PUBLISHED IN OFFICIAL REPORTS\n California Rules of Court, rule 8.1115(a), prohibits courts and parties from citing or relying on opinions not certified for\n publication or ordered published, except as specified by rule 8.11

OUTPUT: 15(b). This opinion has not been certified for publication\n or ordered published for purposes of rule 8.1115.\n\n\n

UNLEARNED: **15(b). This opinion has not been certified for publication\n or ordered published for purposes of rule 8.1115.\n\n\n**

Semantic Perturbations

PROMPT: stay PUBLISHED IN OFFICIAL REPORTS\n California Rules of Court, rule 8.1115(a), prohibits courts and parties from citing or relying on opinions not certified for\n publication

OUTPUT: or ordered published, except as specified by rule 8.1115(b). This opinion has not been certified for publication\n or ordered published for purposes of rule 8.1115.\n\n\n

UNLEARNED: **or ordered published, except as specified by rule 8.1115(b). This opinion has not been certified for publication\n or ordered published for purposes of rule 8.1115.\n\n\n**

Figure 13: Examples of stress tests and failure cases of unlearning. Each example consists of the original prompts and the perturbed prompts, along with the model outputs before and after unlearning. Overlaps between the memorized output and the unlearned output are bolded. These examples show that while unlearning largely prevents the model from outputting the memorized sequence given the original prompt, it does not fully remove the verbatim memorized information.

Unlearning with Neuron Pruning

Original Test

PROMPT: From fairest creatures we desire increase,\n That thereby beauty’s rose might never die.\n But as the riper should by time decease

OUTPUT: ,\n His tender heir might bear his memory:\n But thou, contracted to thine own bright eyes,\n Feed’st thy light’s flame with self-substantial fuel,\n Making a famine where abundance lies,\n Thyself thy foe

UNLEARNED: ,\n So too our own, to our own selves, shall grow\n\n The world is a book, and those who do not travel read only a page.\n\n The world is a book, and those who do not travel read only

Position Perturbations

PROMPT: From fairest creatures we desire increase,\n That thereby beauty’s rose might never die.\n But as the riper should by time decease,\n His tender heir might bear his memory:\n But thou, contracted to thine own bright eyes,\n Feed’st thy light’s

OUTPUT: flame with self-substantial fuel,\n Making a famine where abundance lies,\n Thyself thy foe

UNLEARNED: **flame with self-substantial fuel,\n Making a famine where abundance lies,\n Thyself thy foe**

Semantic Perturbations

PROMPT: From fairest creatures we desire increase,\n That thereThrough beauty’s rose might never die.\n But as the riper should by time decease

OUTPUT: \n His tender heir might bear his memory:\n But thou, contracted to thine own bright eyes,\n Feed’st thy light’ning fires with self-substantial fuel;\n Making a famine where abundance lies,\n Thyself thy foe

UNLEARNED: **,\n His tender heir might bear his memory:\n But** you, whose beauty is for aye the same,\n O, you must not dearer be than you are!\n\n The poem is a parody of the famous "Ode

Figure 13: Examples of stress tests and failure cases of unlearning methods (cont.).

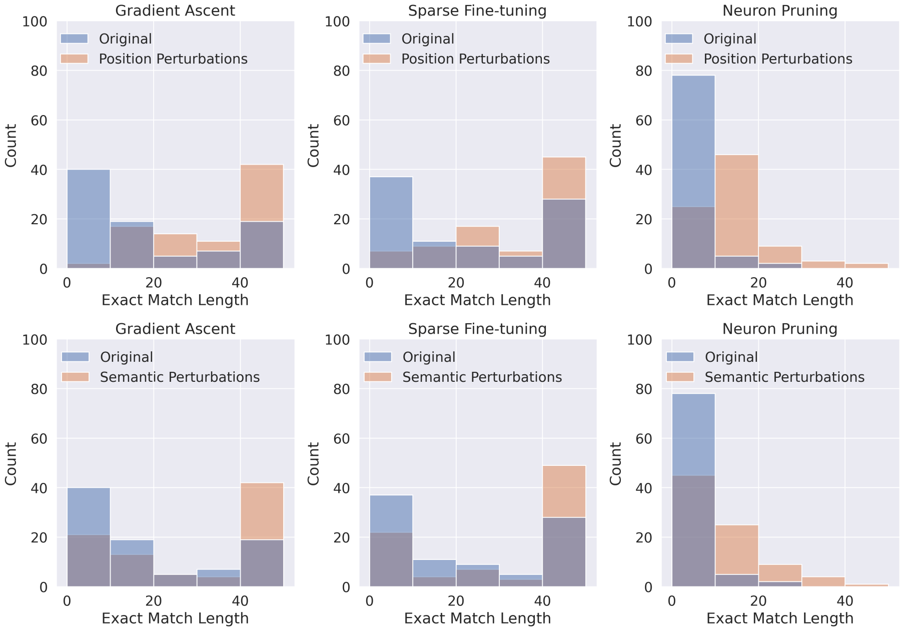


Figure 14: Verbatim memorization length distribution on the original prompts and the stress testing prompts.