Preventing Verbatim Memorization in Language Models Gives a False Sense of Privacy

Anonymous ACL submission

Abstract

Studying data memorization in neural language models helps us understand the risks (e.g., to privacy or copyright) associated with models regurgitating training data and aids in the development of countermeasures. Many prior works—and some recently deployed defenses—focus on "verbatim memorization", defined as a model generation that exactly matches a substring from the training set. We argue that verbatim memorization definitions are too restrictive and fail to capture more subtle forms of memorization. Specifically, we design and implement an efficient defense that *perfectly* prevents all verbatim memorization. And yet, we demonstrate that this "perfect" filter does not prevent the leakage of training data. Indeed, it is easily circumvented by plausible and minimally modified "style-transfer" prompts—and in some cases even the non-modified original prompts—to extract memorized information. We conclude by discussing potential alternative definitions and why defining memorization is a difficult yet crucial open question for neural language models.

1 Introduction

The ability of neural language models to memorize their training data has been studied extensively (Kandpal et al., 2022; Lee et al., 2021; Carlini et al., 2022; Zhang et al., 2021; Thakkar et al., 2021; Ramaswamy et al., 2020). When language models, especially ones used in production systems, are susceptible to *data extraction* attacks, it can lead to practical problems ranging from privacy risks to copyright concerns. For example, Carlini et al. (2021) showed that the GPT-2 language model could output personally identifying information of individuals contained in the training dataset.

One natural way to avoid this risk is to filter out any generations which copy long strings verbatim from the training set. GitHub's Copilot, a language-model-based code assistant, deploys this defense by giving users the option to "block suggestions matching public code" (GitHub, 2022).

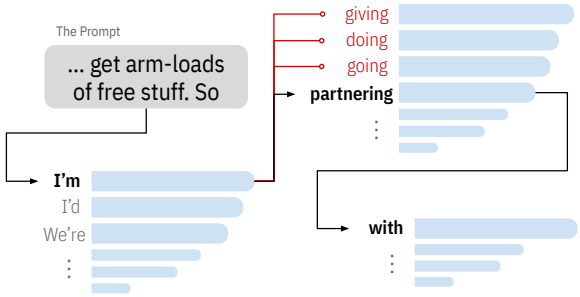


Figure 1: Illustration of Memorization-free Decoding, a defense which can eliminate verbatim memorization in the generations from a large neural language model, but does not prevent approximate memorization.

In this work, we ask the question: "*Do language models emit paraphrased memorized content?*" This scenario can happen maliciously (e.g., adversaries trying to extract private user data) or through honest interactions (e.g., users prompting in real-world scenarios). Indeed, we find that Copilot's filtering system is easy to circumvent by applying plausible "style transfers" to the prompt. For example, by translating variable names from English to French the model outputs completely memorized examples, but post-processed with the en-fr style transfer. We further show that GPT-3 (Brown et al., 2020), a model trained on natural language, is also vulnerable to extraction attacks.

Unfortunately, Copilot's training set and precise algorithm for their defense are non-public. Therefore, to investigate this phenomenon systematically, we develop MEMFREE decoding (Figure 1), an efficient defense that is guaranteed to prevent all verbatim memorization, and which scales to training sets consisting of hundreds of gigabytes of text. In MEMFREE decoding, at each step of generation we check whether the model's chosen next token would create an n-gram found in the training set. If it does, an alternative next token is selected (without a computationally expensive regeneration) by

sampling from the model's token posterior. The check for membership in the training set is performed efficiently using a Bloom filter containing all common n-grams from the training set.

We use MEMFREE to study Copilot's verbatim-filtering defense on other state-of-the-art large language models such as GPT-Neo (Gao et al., 2020). We first confirm that even honestly designed prompts often bypass verbatim memorization checks. Then, we observe another interesting phenomenon: language models succeed at emitting *approximate memorization* that bypass our filter all by themselves. Indeed, when prevented from generating exact n-grams from the training set, models are capable of "cheating" the filter by producing close paraphrases–for example, inserting spelling errors, adjusting punctuation or whitespace, or using synonyms (e.g., swapping 'and' with '&'). These changes lead to generated text a human would perceive as nearly identical, even if it is not verbatim memorization.

Clearly, defenses which prevent verbatim copying are *necessary but not sufficient* to protect against training data leakage. As a result of these failure modes, we argue that a broader definition of memorization is necessary when reasoning about training set memorization in language models. Such a definition should not only capture verbatim notions of memorization, but also notions based on high "semantic similarity" between model outputs and training data. We conclude our work by comparing approximate and verbatim memorization, discussing their relation to other domains of literature, and the challenges surrounding the ambiguity of approximate memorizations. Future work that aims to faithfully measure or prevent memorization in language models will need to take this ambiguity into account—for example, our analysis suggests that the fraction of datasets that large language models is likely far larger than the fraction as reported in prior work (Carlini et al., 2022).

2 Background

Language Models. We consider auto-regressive language models that operate over sequences of text and, given a prefix p, output a probability distribution for the next token in the sequence. To generate text for a prompt p, the language model starts with an empty suffix s, and repeatedly samples the next token from its prediction on $p + s$, and then appends this token to s. The success of neural language models has, in large part, been driven by the transformer architecture introduced of Vaswani et al. (2017), which allowed models to scale from millions to hundred of billions of parameters over the past half-decade (Brown et al., 2020; Chowdhery et al., 2022; Zhang et al., 2022). This increase in model sizes has likewise driven increases in dataset sizes, with most of this data coming from internet crawls (Lee et al., 2021; Raffel et al., 2020; Gao et al., 2020).[1]

Prior work has shown that large language models can memorize and regurgitate potentially private information, like phone numbers and addresses, as well as memorize long sequences from their training sets (Carlini et al., 2019, 2021; Lee et al., 2021; Carlini et al., 2022; Zhang et al., 2021; Thakkar et al., 2021; Ramaswamy et al., 2020; Kandpal et al., 2022). Our work focuses on large language models trained to generate English text or code.

Measuring Memorization. Many studies of memorization stem from a concern of privacy leakage: if a model memorizes sensitive training data and can generate it, then interactions with a model can lead to the leakage of that sensitive data. Nearly all of this literature is focused on measuring *verbatim cases of memorization*.

Eidetic memorization (Carlini et al., 2021) defines a string s as memorized if there exists a prompt p so that $f(p) = s$ and s is contained in the training dataset. This definition and variations of it have been used widely in the literature (Kandpal et al., 2022; Lee et al., 2021; Carlini et al., 2022). For example, Tirumala et al. (2022) study a similar per-token definition called *exact memorization* and Kandpal et al. (2022) a document-level definition called *perfect memorization*.

There is also a newly emerging line of works exploring *differential-privacy (DP)-based definitions* (Zhao et al., 2022; Stock et al., 2022), which relate to document-level DP guarantees in language modelling (Yu et al., 2021). These works differ from the above in that they define a probabilistic leakage measure. However, this is based on the probability of generating—verbatim—a canary sentence s, depending on whether s was contained in the training set or not. There are different probabilistic definitions, also based on verbatim sequences, such as the *counterfactual memorization* proposed by Zhang et al. (2021).

[1] A common source for datasets is the Common Crawl dataset found at: https://commoncrawl.org/

In the domain of language model memorization, the most similar work to ours is Lee et al. (2021) who also argue for a more relaxed definition of memorization. Lee *et al.* say any model output for a prompt p is memorized if it is within some chosen edit distance of the prompt's true continuation in the training set. As we will discuss, a small edit distance may not capture all forms of approximate memorization either—such as our examples of "style-transfer" applied to memorized content.

Preventing Memorization. Differentially private training, e.g., using DP stochastic gradient descent (Abadi et al., 2016), is the gold standard for training models which provably do not memorize individual training examples. However, in practice, these techniques result in worse generative models (Anil et al., 2021)—thus, no state-of-the-art, large, language models are trained with DP. Instead, data deduplication has arisen as a pragmatic countermeasure against data memorization (Lee et al., 2021; Kandpal et al., 2022; Carlini et al., 2022). The core idea is to remove any duplicated content—e.g., repeated documents—because duplicated content is much more likely to be memorized. However, deduplication does not *guarantee* that a model will not still memorize individual (deduplicated) examples, necessitating defenses that operate at inference-time.

3 Preventing Models from Emitting Verbatim Training Data

In this paper, we consider inference-time defenses that eliminate the generation of memorized content from the training set. The most immediate way to do this is simply to filter all model outputs using some fixed definition of memorization. For example, in Carlini et al. (2022), a continuation $s = f(p)$ of a k-length prompt p is said to be memorized if the string s exists verbatim in the training dataset. A straightforward implementation checks each generation s against the training set and rejects any matches. We call the approach of re-running a language model, possibly many times with different seeds, until a qualifying generation is produced, **retroactive censoring**.

The problem with retroactive censoring is that it effectively prevents the model from emitting any output when the model's confidence in a memorized string is too high. To encourage a model to generate novel outputs, we could also adopt a more granular filtering approach: rather than censoring memorized content solely at the level of an entire sequence s, we could instead check and mark each n-gram within s individually. Filtering for memorization at the n-gram-level rather than at the sequence level allows substrings of a generation which may be novel to be kept, and only the pieces that are verbatim memorized to be modified. We call this approach **MEMFREE decoding**, as the defense is applied at decoding time.

Both retroactive censoring and MEMFREE decoding explicitly prohibit the model from emitting a sequence if it is contained (entirely or partially) in the training dataset. However, in retroactive censoring, if a generation starts off with memorized text, but then veers off track from the true continuation (a common occurrence), this would not be marked as memorization, even though a portion of the output sequence is clearly memorized. The MEMFREE decoding approach performs a more fine-grained and aggressive check by filtering out all memorized subsequences of a given length. In this work we use the MEMFREE decoding approach to show that even when a model is restricted from emitting any output with snippets of verbatim memorization, the model can still leak training data.

3.1 MEMFREE Decoding Details

In order to implement MEMFREE decoding, we alter the model's generation in an online manner by restricting the production of tokens which would result in an n-gram memorization. Let p be the current working prefix and t be the next proposed token when running the model forward.

Our algorithm first checks if any n-gram in the concatenated sequence $p||t$ is contained in the training dataset D. If it is, we suppress this generated token and re-sample from the model. To avoid potentially expensive resamplings, we equivalently express this as altering the model's output probability distribution by removing the probability mass from token t. In this way, we guarantee that prior to sampling the probability of outputting a memorization will be 0. Appendix B.1 gives a formal procedure for this method.

Altering the token posterior allows any sampling strategy to be used on top of memorization-free decoding. For example, if one uses top-k sampling, tokens that result in memorization are disqualified before the probability distribution is truncated to the k next most likely tokens.This procedure is guaranteed to generate non-memorized text.

3.2 Querying the Training Set Efficiently

Our MEMFREE defense has assumed that it is easy to perform the query $s \in D$ to test if any given string is contained in the training dataset. Because the defense works at inference-time, it is necessary that this query is computationally efficient to maintain utility of the language model. Given that training sets may contain terabytes of data (Brown et al., 2020), it is infeasible to maintain an entire copy of the training dataset in an efficiently accessible storage. Thus, we explore three optimizations to speed up the process of memorization checking.

First, as a direct result of our n-gram memorization definition, we can equivalently check only the n-gram ending in the current predicted token t; we can thus avoid many n-gram queries for each token. Further, and in addition to preventing subsequence memorization, this allows us to avoid queries into a large set of all prefixes and continuations.

Second, we only check against sequences that have a reasonable probability of being memorized by the model. In theory, this could be easily determined by running each n-gram $s \in \mathcal{D}$ through the model and then filtering out all sequences with high loss (thus unlikely to be memorized). However, this is a computationally expensive procedure as it requires re-processing every substring of the training dataset. Instead, a computationally- and storage-efficient procedure could be to only store n-grams which occur more than once in the training set—prior work has shown duplicate text is the most likely to be memorized (Lee et al., 2021; Kandpal et al., 2022).

Third, by being willing to tolerate some false positives (labeling an n-gram as memorized when it is in fact not), we can take advantage of probabilistic data structures such as Bloom filters (Bloom, 1970), which admits no false negatives but trades off time and space with the false positive rate (which can be computed exactly). Thus, by using a Bloom Filter, we guarantee that no memorized n-gram will ever be released (i.e., a false negative) but we may (rarely) prevent the emission of non-memorized content (i.e., a false positive).

Integrating a Bloom Filter into our defense is straightforward. Let $\mathcal{F}_{fp}(\mathcal{D}_n)$ represent the Bloom Filter of dataset $\mathcal{D}$, generated by adding each n-gram of the dataset $s \in \mathcal{D}_n$ to the Bloom filter, with false positive rate fp. Then, any memorization check $s \in \mathcal{D}_n$ in Algorithm 1 can be replaced with $s \in \mathcal{F}_{fp}(\mathcal{D}_n)$. The Bloom filter can be generated with a single pass over the model's training set, which could be performed in parallel with one epoch of model training.

Additional Parameters. We must choose an appropriate false positive rate based on memory constraints and the chosen n-gram length. Choosing n has two major impacts: on the population size (i.e., the number of unique n-grams) and thus the size of the filter, and on the effectiveness of memorization mitigation. If n is set too low, then we will certainly prevent all memorized sequences but might also prevent too many common phrases. But if we set n too high, we might not prevent actually memorized sequences from being emitted by the model. We discuss these tensions in Appendix B, along with two additional takeaways: (1) that MEMFREE *does not impact downstream model performance* (which may result from false positives), and (2) that our chosen optimizations maintain a suitably low false negative rate (we observed a 3000x improvement). These optimizations led to a filter of size 1.6 gigabytes (or, 40.5 gigabytes if all, even non-duplicated, 10-grams were stored) when run over the 800GB Pile dataset.

3.3 Measuring Approximate Memorization

To show that defenses against verbatim memorization still allow approximate memorization, we need a definition for approximate memorization. We consider two definitions. First, drawing from standard NLP evaluation techniques, we measure the BLEU score between the generated and ground-truth continuations. Second, we measure the length-normalized character-level Levenshtein similarity between the generated and ground-truth continuations. Appendix C.1 gives implementation details. In Section 5, we investigates how these two similarity metrics decrease with MEMFREE decoding.

For situations requiring a binary label of whether approximate memorization has occurred, we use the following definition: a suffix s for prefix p is labeled as memorized if for generation $g = f(p)$, $\text{BLEU}(g, s) > 0.75$. This threshold was chosen by qualitatively inspecting examples. Several example generations that are close to this threshold are shown in Table A11.

When we repeat the prefix-extraction experiment from (Carlini et al., 2022) to measure incidents of generations that could be considered memorized, but using this approximate definition instead of a verbatim one, we find that hat prior literature has

Standard prompting with original prefix and format

```
float Q_rsqrt( float number )
{
long i;
float x2, y;
const float threehalfs = 1.5F;

x2 = number * 0.5F;
y  = number;
i  = * ( long * ) &y;
Copilot no longer generates continuations
```

Prompt with Python-style comment

```
# float Q_rsqrt( float number )
# {
# long i;
# float x2, y;
# const float threehalfs = 1.5F;
#
# x2 = number * 0.5F;
# y  = number;
# i  = * ( long * ) &y;
# i  = 0x5f3759df - ( i >> 1 );
# y  = * ( float * ) &i;
# y  = y * (threehalfs - (x2*y*y));
#
# return y;
#}
```

Prompt with French translation (alternate naming convention)

```
float Q_sqrt( float nombre )
{
long i;
float x2, y;
const float trois_moitie = 1.5F;

x2 = nombre * 0.5F;
y  = nombre;
i  = * ( long * ) &y;
i  = 0x5f3759df - ( i >> 1 )
y  = * ( float * ) &i;
y  = y * (trois_moitie - (x2*y*y));
//y  = y * (trois_moitie - (x2*y*y));

return nombre * y;
}
```

Figure 2: **Honest "style-transfer" prompts evade verbatim memorization filters.** Trivially modifying prompts causes GitHub's Copilot language model to emit memorized, but not verbatim, content. Prompts highlighted in blue. Model evaluated with the option "block suggestions matching public code" enabled. For brevity, we removed comments from model outputs.

significantly underestimated memorization leakage. In Figure 3, the shaded region represents the fraction of memorized samples that would have bypassed a verbatim memorization filter: in the worst case, there is a factor-of-two increase.

However, we caution that this definition of approximate memorization is inaccurate, potentially both over and under counting approximate memorization. While our choice of a 0.75 BLEU score threshold shows a significant increase in approximate vs. verbatim memorization, it is not clear that all identified cases of memorization would be perceptually tagged as such by a human judge. This is one reason why simply switching to this definition for defenses may not be ideal—it could introduce

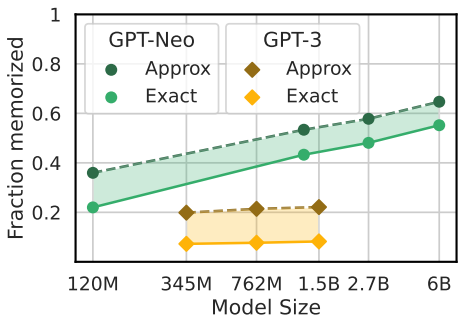


Figure 3: **Significantly more examples are approximately memorized (BLEU > 0.75) than are found to be exactly memorized by Carlini et al. (2022)**. This is for undefended generation.

significant false positives.

4 Evading Verbatim Mem. Defenses

In this section, we show how retroactive censoring of verbatim memorization can be evaded, even in settings where models are used honestly. We first present a case study with Copilot, which has implemented retroactive censoring in production. We then show how a large English language models like GPT-3 are susceptible to the same vulnerability, should a defense similar to Copilot's be deployed. In short, protecting against verbatim memorization can lead to a false sense of privacy.

4.1 Evading Copilot's Memorization Filter

Copilot is a code auto-complete service which is trained on GitHub code. Copilot is built using the Codex language model designed by OpenAI (Chen et al., 2021). To prevent generating memorized code, Copilot uses a filtering mechanism that blocks model outputs from being suggested if they overlap significantly (approximately 150 characters) with a training example. This is a practical example of a filter that aims at preventing perfect verbatim memorization, presumably by using a procedure similar to Algorithm 1 (the exact mechanism used by GitHub is not public). However, we find that the filter fails to prevent the leakage of training data in many settings.

Style-transfer prompting. In Figure 2, we show that Copilot's filter can easily be bypassed by prompts that apply various forms of "style-transfer" to model outputs, thereby causing the model to produce memorized (but not verbatim) outputs.

As a concrete example, we demonstrate how to extract the public code for Quake's "Fast Inverse

Square Root". If we naively prompt the model with the function definition `"float Q_rsqrt ( float number )"`, Copilot correctly aborts generation of the full function ("standard prompting").

However, we find that simple style-transfers applied to the prompt allow us to easily bypass Copilot's restrictions. First, via prompting with "Python-style comments" we begin our prompt with Python's comment character "#". Even though this is syntactically invalid C code, Copilot outputs the entire verbatim fast inverse square root algorithm, but commented out. Second, in prompting with "French translations" we change the naming convention to French. As a result, the generations follow the new naming convention and are no longer flagged as a verbatim match. Other naming conventions, such as pre-pending "_" to the variable or changing the language to Spanish, also work.

These strategies work because the Copilot model is sufficiently powerful: it can both follow the style-transfer prompt (by e.g., renaming variables) while simultaneously regurgitating memorized training data. We provide more examples in Appendix E.

Copilot Evades its own Filter Not only do *actively* style-transfered prompts evade the verbatim memorization filter, but even passively prompting Copilot with highly duplicated text from the Pile dataset can too. We find several examples where Copilot evades its own filter to output memorized text, some of which we show in Figure 5. We see that Copilot evades the filter by (1) changing capitalization, (2) making small non-stylistic errors, and (3) changing whitespaces. The latter evasion (changing whitespaces) is surprising, as Copilot's documentation reports ignoring whitespace in its filtering mechanism (Appendix A). However, we hypothesize that this can be explained by the model replacing tabs with space characters. We can verify this by adding tabs to the beginning of each line of the Q_sqrt function, as an application of our style-transfer strategy.

4.2 English Language Models

Following our analysis of Copilot, we ask whether this vulnerability is pervasive is other language models too. We use API access to three versions of GPT-3 Davinci to test whether they would be susceptible to style transfer of the prompt. Since the training set is unknown, we prompt with documents we believe are likely to have been memorized: open-source licenses, famous speeches and monologues, novel openings, and song lyrics.For each document, we prompt the model with 100 words of either (1) the original document ("base"), (2) the document with all spaces doubled ("spaces"), (3) the document in all lowercase ("lower"), and (4) the document in all uppercase ("caps"). We report approximate memorization results of this experiment in Figure 4, with additional figures in Appendix D.

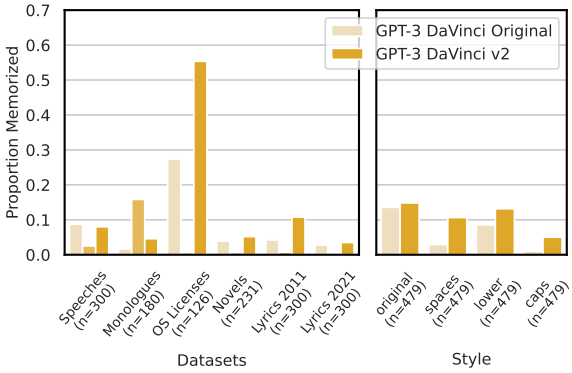


Figure 4: Fraction of prompts which discover approximate memorization, grouped by domain (left) and by style transfer applied (right). Full plot in Appendix D.

We see that even when prompting with style-transfered prompts, GPT-3 is still often able to generate a memorized continuation. Defenses for verbatim memorization are therefore incomplete. Among the three techniques, uppercasing was the least likely to lead to memorized generations. It is also interesting to note that different versions of GPT-3 have wildly different memorization tendencies, emphasizing the importance of models' training set compositions and training methods.

5 MemFree Decoding Experiments

In this section, we study the effectiveness of our proposed MemFree decoding defense from Section 3.3, and study the appropriateness of our proposed definition of approximate memorization.

Experimental Design It is not possible to apply MemFree to the models from Section 4 since their training sets are non-public. Instead, we turn to the GPT-Neo languge model family (Black et al., 2021). These models are trained on the Pile, a publicly available 825GB dataset (Gao et al., 2020). We build a Bloom filter over all 10-grams occur 10 or more times (more details in Appendix B). In all experiments, we generate text using $\arg\max$ decoding as the sampling method. We investigate

1) Misspelling and changed capitalization

```
This program is free software"; you can redistributeredistribut it and/or *
modify it under the terms of the GNU General Public License
* asAS published by the Free Software Foundation; either
version 2 * of the License, orOR (at your option) any later
version"
```

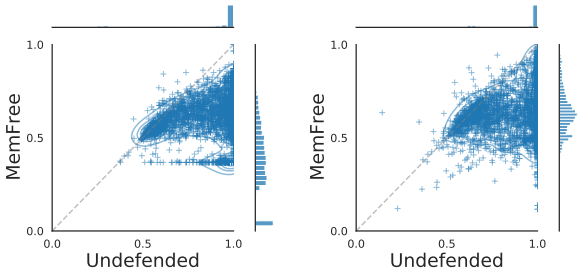
2) Small non-stylistic errors

```
@aws-sdk/protocol-http";
import { Command as $Command } from "@aws-sdk/smithy-client";
import {FinalizeHandlerArguments,Handler,HandlerExecutionContext,
MiddlewareStack,HttpHandlerOptions} as __HttpHandlerOptions,MetadataBearer
as __MetadataBearerMetadataBear,
```

3) Changed whitespace

```
[...]IPV6_2292PKTINFO(2 ws)\t\t\t\t\t = 0x2\nIPV6_2292PKTOPTIONS \t\t\t\t = 0x6\n
IPV6_2292RTHDR [20 spaces][9 spaces]= 0x5\n
```

Figure 5: **CoPilot can "cheat" and emit nearly verbatim memorized content.** Here, we show prompts from the training set, where the model makes slight errors causing the continuations to pass the filter. Prompts are in cyan, followed by CoPilot's continuation where errors are highlighted as **model's generation in orange** with the **correct characters in green**.



(a) BLEU (word-level) (b) Edit similarity (char-level)

Figure 6: **MemFree reduces similarity when the continuation would have been highly similar to the ground-truth**, and has little impact otherwise. For 5,000 prompts, we plot the similarity of the groundtruth continuation with the generation from MemFree (y-axis) and with the undefended generation (x-axis). Generations on the diagonal were not memorized.

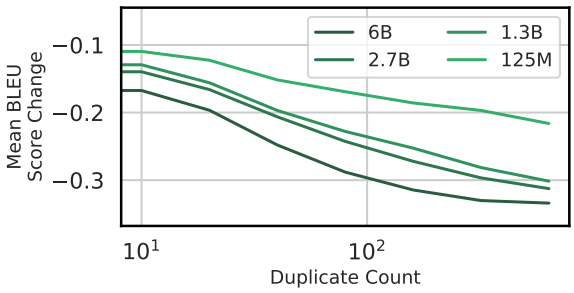


Figure 7: **MemFree decreases the BLEU score of generations more for highly duplicated examples**.

four model sizes: 125M–6B parameters.

We evaluate using substrings of the Pile released by Carlini et al. (2022). The dataset includes 30k strings of length 150 tokens taken from the training set. These are divided into 30 buckets of 1k strings, sampled such that the strings in bucket i occur in the Pile between $2^{i/4}$ and $2^{(i+1)/4}$ times. For each string, we use the first 50 tokens as a prompt p and generate a 50-token long continuation.

Reduction in Memorization MemFree significantly reduces the similarity of generations to the groundtruth, compared to performing undefended generation (Figure 6). We also observe that when undefended generation already results in low similarity with the groundtruth, MemFree does not significantly alter the generations, as desired.

Previous work shows that increasing model size increases discoverable memorization (Carlini et al., 2022; Kandpal et al., 2022). We again find a clear trend that generations from larger models have, on average, a much higher similarity with the original continuation (Figure 8). Despite this, MemFree remains effective at all model sizes (BLEU remains near-flat around 0.6). Even when a sequence has many duplicates in the train set (a strong indicator of memorization), MemFree significantly decreases similarity with the groundtruth at all model sizes (Figure 7).

Failures in Preventing Memorization A defense against memorization fails when it allows a sequence to be generated which a human would perceive as substantially copied from the true continuation—even if it is not verbatim memorized. This failure case can be seen as the points where the MemFree generation is still a close match to the ground-truth continuation (Figure 6). It occurs because the defense only adjusted a few tokens (e.g., 1 after every sequence of 10). When looking at these examples, many, but not all, are lists of numbers. Some examples are included in Table A14. There is also a second failure-case: when a full (50 token) generation is made more similar with the ground-truth by MemFree (on 10-grams) than without. This may happen depending on the model's token posterior's after removing all tokens that fail the MemFree check. Almost all of these cases had a trivial increase in similarity. However, 0.16% of samples had a similarity increase above 0.1. We found qualitatively that many of these cases did have significant overlap with the true continuation.

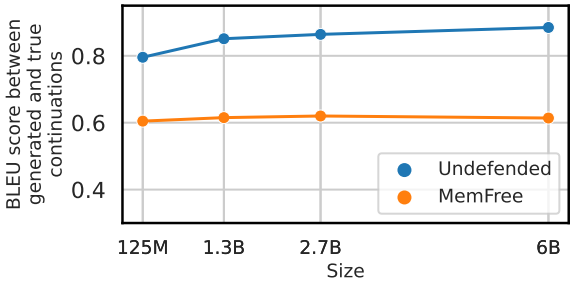


Figure 8: **MEMFREE remains effective at reducing similarity between the generated and groundtruth continuations even as models grow larger.**,

6 Discussion

Defining Memorization in Language Models. While verbatim definitions have helped discover significant memorization in large language models, they are insufficient to capture more subtle forms of memorization. Our work highlights two such situations: "style-transfer" prompting, where defenses for verbatim memorization can be actively subverted, and when models "cheat" by outputting similar, but not verbatim, continuations. As a result, our work suggests that memorization prevention must capture these types of paraphrased memorizations in addition to the previously considered verbatim definitions. However, exhaustively anticipating styles to incorporate into defenses is an innumerable problem that will become harder as models become more powerful.

This emphasizes two major challenges in defining approximate memorization. First, new approximate cases must be discoverable by the definition which can result in some cat-and-mouse game. Second, this definition is domain-dependent. For example, our paper focuses on language models trained to output English and code—other languages will require different considerations when defining memorization.

There are two areas of research which may help in improving memorization definitions. The field of **image generation memorization** is already comfortable with measuring *fuzzy* (in ours, *approximate*) memorization, where generated items may be perceptually similar to training set examples, despite having high distance according to standard metrics. For example, Fredrikson et al. (2015) consider "model inversion", where an image is successfully recovered from the model if it is identifiable to a human worker. In Zhang et al. (2020), model inversion success is measured based on pixel similarity and feature space similarity to training images. These works also recover "representative" images from different classes, rather than specific training examples. Recent work on reconstructing training images have used feature similarity (Haim et al., 2022) and pixel similarity (Balle et al., 2022). In each of these papers, "fuzzy" reconstructions are allowed by the evaluation metrics and, indeed, are common in their reconstructions.

The inherent limitations of verbatim definitions of text regurgitation have also been well documented in the literature on **plagiarism detection**—both for text and code. Existing plagiarism tools, and their evaluations, go far beyond verbatim matches and consider fuzzy data "clones" ranging from simple transformations (e.g., word variations or shuffles) to arbitrary semantics-preserving paraphrasing (Roy et al., 2009; Potthast et al., 2010). Re-purposing techniques from the plagiarism detection literature to minimize generation of memorized data in LLMs is an interesting direction toward achieving better approximate memorization definitions in machine learning.

Consequences for machine learning research. In relaxing definitions of memorization, our paper acknowledges the blurred line between memorization (e.g., of personal information) and knowledge (e.g., of common facts). Because we use a 10-gram overlap, our MEMFREE decoding algorithm should not significantly impact utility, however studying this interplay is an important area of future work. However, still, identifying which data is considered "memorized" cannot be done only by looking for verbatim reproductions of the training set. This may make the task of understanding memorization and generalization more difficult.

We do not believe that our work requires abandoning all research directions which rely on prior verbatim definitions. These definitions are still useful as an efficient way to test for obvious and undeniable memorization. However it will be necessary to continue studying further relaxations of memorization definitions to adequately capture and measure the space of privacy concerns for language models.

7 Ethics & Broader Impact

Improving the privacy of neural language models—and especially those trained on user data—is an important and timely research problem. In this paper we hope to help both researchers and practi-

tioners develop a more nuanced understanding of what constitutes memorization in language models. In particular, just because a sequence does not appear verbatim in a training dataset does not mean the example is a novel generation: as we have shown, models today are sufficiently powerful to minimally transform memorized data to make it appear superficially different even if the underlying content remains memorized.

Our observation will complicate the privacy evaluation of future machine learning models. It should no longer be deemed sufficient to check for (verbatim) matches between generated output and a training example. Practitioners in the future will need to be aware of this potential failure mode when applying output post-processing defenses to mitigate memorization. To the best of our knowledge, the only deployed system affected by our analysis is GitHub's Copilot. In order to mitigate harm here we shared a copy of our paper with the relevant researchers at both GitHub and OpenAI prior to paper submission.

In this paper we focus our efforts entirely on *public* datasets that other researchers have extensively studied (Gao et al., 2020) to minimize any harm caused by demonstrating extraction results. However, just because the data that we study is public does not mean there are no privacy concerns. As Brown et al. (2022) argue, there are many other considerations when discussing the privacy of large models trained on "public" datasets.

References

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

Rohan Anil, Badih Ghazi, Vineet Gupta, Ravi Kumar, and Pasin Manurangsi. 2021. Large-scale differentially private BERT. *arXiv preprint arXiv:2108.01624*.

Borja Balle, Giovanni Cherubin, and Jamie Hayes. 2022. Reconstructing training data with informed adversaries. *arXiv preprint arXiv:2201.04845*.

Michael A Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. 2018. Bloom filters, adaptivity, and the dictionary problem. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–193. IEEE.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow. *If you use this software, please cite it using these metadata*, 58.

Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Hannah Brown, Katherine Lee, Fatemehsadat Mireshghallah, Reza Shokri, and Florian Tramèr. 2022. What does it mean for a language model to preserve privacy? Seoul, Korean. ACM Conference on Fairness, Accountability, and Transparency.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Jehoshua Bruck, Jie Gao, and Anxiao Jiang. 2006. Weighted Bloom filter. In *2006 IEEE International Symposium on Information Theory*, pages 2304–2308. IEEE.

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

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.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333.

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

GitHub. 2022. About GitHub Copilot. https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. 2022. Reconstructing training data from trained neural networks. *arXiv preprint arXiv:2206.07758*.

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

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2021. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*.

Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. 2010. An evaluation framework for plagiarism detection. In *Coling 2010: Posters*, pages 997–1005.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Swaroop Ramaswamy, Om Thakkar, Rajiv Mathews, Galen Andrew, H Brendan McMahan, and Françoise Beaufays. 2020. Training production language models without memorizing user data. *arXiv preprint arXiv:2009.10031*.

Chanchal K Roy, James R Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming*, 74(7):470–495.

Pierre Stock, Igor Shilov, Ilya Mironov, and Alexandre Sablayrolles. 2022. Defending against reconstruction attacks with Rényi differential privacy. *arXiv preprint arXiv:2202.07623*.

Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. 2011. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155.

Om Dipakbhai Thakkar, Swaroop Ramaswamy, Rajiv Mathews, and Francoise Beaufays. 2021. Understanding unintended memorization in language models under federated learning. In *Proceedings of the Third Workshop on Privacy in Natural Language Processing*, pages 1–10.

Kushal Tirumala, Aram H Markosyan, Luke Zettlemoyer, and Armen Aghajanyan. 2022. Memorization without overfitting: Analyzing the training dynamics of large language models. *arXiv preprint arXiv:2205.10770*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. 2021. Differentially private fine-tuning of language models. *arXiv preprint arXiv:2110.06500*.

Chiyuan Zhang, Daphne Ippolito, Katherine Lee, Matthew Jagielski, Florian Tramèr, and Nicholas Carlini. 2021. Counterfactual memorization in neural language models. *arXiv preprint arXiv:2112.12938*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. 2020. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 253–261.

Xuandong Zhao, Lei Li, and Yu-Xiang Wang. 2022. Provably confidential language modelling. *arXiv preprint arXiv:2205.01863*.

A GitHub Copilot

At the time of this paper's writing, GitHub Copilot's memorization prevention mechanism is described in their FAQ at https://github.com/features/copilot. We copy the text here:

> **What can I do to reduce GitHub Copilot's suggestion of code that matches public code?**
>
> We built a filter to help detect and suppress the rare instances where a GitHub Copilot suggestion contains code that matches public code on GitHub. You have the choice to turn that filter on or off during setup. With the filter on, GitHub Copilot checks code suggestions with its surrounding code for matches or near matches (ignoring whitespace) against public code on GitHub of about 150 characters. If there is a match, the suggestion will not be shown to you. We plan on continuing to evolve this approach and welcome feedback and comment.

B Further Discussion of MEMFREE

B.1 Formal Procedure

Algorithm 1 provides a formal procedure for MEMFREE decoding. In all our experiments, we used $\arg\max$ decoding as the sampling method for line 4.

Algorithm 1 MEMFREE decoding algorithm.

1: **procedure** GREEDY MEMFREE DECODING(model f, prefix p, gen length n, training set D)
2: **repeat**
3: logits $\leftarrow f(p) - \infty \cdot \{\mathbb{1}[(p||t) \in D] : t \in \text{vocab}\}$
4: tok $\leftarrow$ sample from logits
5: $p \leftarrow p||\text{tok}$
6: **until** n iterations
7: **end procedure**

B.2 Choice of n-gram length

Choosing the n-gram length has two main tradeoffs: it changes the granularity of the memorization checking and the total number of substrings of the dataset that must be stored in the Bloom filter. On the former, notice that short n-grams do not have sufficient novelty (loosely, entropy) to be considered memorizations, e.g., they are often commons words and phrases. However, too large also would not capture shorter sequences that have sufficient novelty. On the latter, notice that the universe of possible n-grams is exponential in n, but that the unique number of such sequences in a fixed dataset may decrease with large n. This total number of unique sequences impacts the required size of the Bloom filter to maintain a fixed false positive rate. With N the number of unique n-grams and fp a decimal probability of the false positive rate, the size of the filter in bits is:

$$m = ceil\left(\frac{-\left(N * \log\left(fp\right)\right)}{\log\left(2\right)^2}\right).$$

Then, k the number of Bloom hash functions can be calculated from the number of bits per element, i.e., m/N, as:

$$k = ceil\left((m/N) * log(2)\right).$$

This determines the cost of inserting and looking up into the Bloom filter as $\mathcal{O}(k)$. But, because k typically remains small (in our case, $k = 7$), this can be treated as a small constant-time operation. See Tarkoma et al. (2011) for the full calculations, which the ones listed here are taken from.

We err on the side of caution and select n=10 for our experiments. This does prevent the model from generating common words or phrases which consist of 10 or more tokens, such as "The quick brown fox

jumped over the lazy dog." or "supercalifragilisticexpialidocious". We find qualitatively that the impact of this is low, and that this also presents a balanced trade-off with the Bloom filter size.

B.3 Choice of Minimum Frequency

Ideally, we want n large enough so that we do not prevent common phrases and small enough so that we catch all (though practically, most) possible memorizations. Optimizing n for this task is both non-trivial, as the objective is not clear, and computationally expensive. Instead, we choose $n = 10$ based on qualitative experience that this does not prevent many common phrases. Further, we do so to also limit the storage cost of the Bloom filter, because n too large leads to a blow up in the number of elements, N.

B.4 Performance of MemFree

In this section, we study two questions: (1) "does MemFree maintain model utility?" and (2) "does our optimized MemFree prevent memorization release".

Along question (1), recall that MemFree can admit false positives, which may degrade the utility of the language model. Fortunately, the false positive rate can be computed exactly, e.g., see Tarkoma et al. (2011), and a long literature has proposed optimizations to account for non-uniform distributions (Bruck et al., 2006) and to adaptively correct for false positives (Bender et al., 2018).

Here, we study how, under reasonable computational constraints and inference times, the observed rates impact model utility. As we will show, we observe that MemFree maintains the highest utility (no observable impact) while being the most efficient defense.

Along question (2), we study if our optimizations lead to a substantial increase in the false negative rate. To do this, we repeat the experiment from (Carlini et al., 2022), which prompted GPT-Neo models with examples from its training data. We compute how many examples are verbatim memorized when MemFree decoding is used. The 6B parameter GPT-Neo model memorizes more than 12,000 of these documents, but, after applying MemFree, it only outputs 4 verbatim memorizations. These 4 remaining verbatim memorizations are repeated fewer than 10 times in the training data, and so were not added to our Bloom filter. Nonetheless, this strategy reduced verbatim memorization by over $3000\times$.

B.5 Bloom Filter Statistics

Figure 9 shows the distribution in number of tokens (out of 50 generated) that were changed by MemFree from the token that would have been generated using undefended greedy decoding.

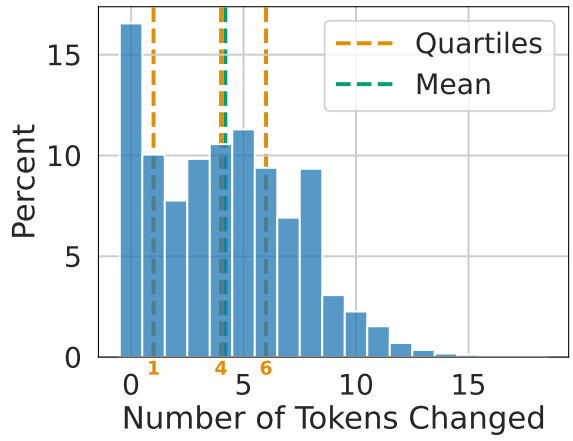


Figure 9: **Most generations require few (< 5) changes to pass MemFreechecks.** Data for histogram from 6000, 50-token generations using MemFree decoding on GPT-Neo 6b.

Figure 10 presented some of the query patterns of the MemFree decoder to investigate when and how it impacts decoding. First, we observe that MemFree is trivial to run in terms of compute: it takes only 49.8 milliseconds to run 10,000 queries on one CPU core. From Figure 10 (left), all generations required significantly fewer queries (mean $= 42.1$ queries / generation)—even running batches of many hundreds

or thousands of queries would incur less than a few seconds additional overhead. Second, we find that the Bloom filter is often hit at the first and tenth tokens after the prompt. We see many hits at the first token because all our prompts are from the training data—so there are relatively fewer single token additions that generate a novel n-gram. Third, we find that most generations need only a few (< 5) alterations due to MEMFREE decoding.

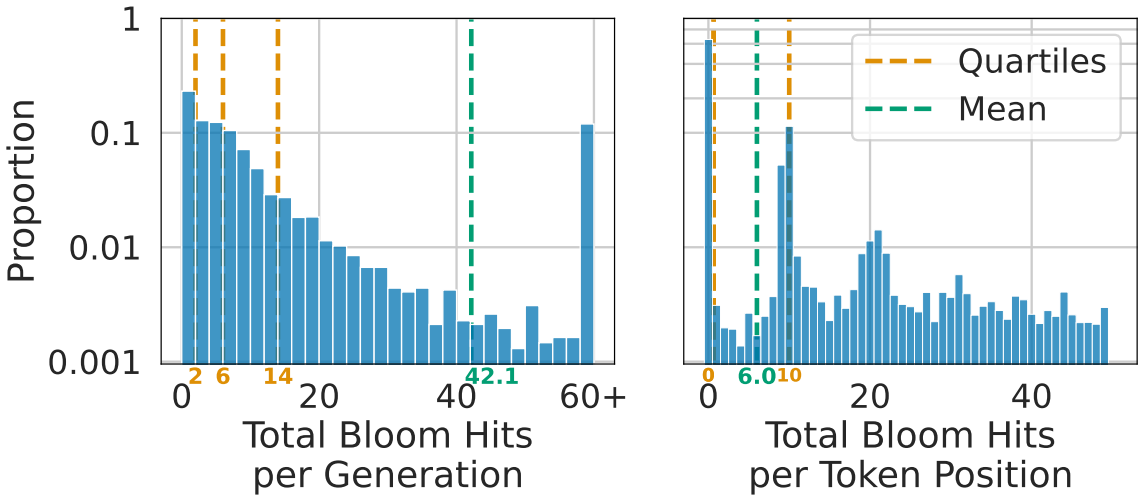


Figure 10: **(left) Most generations have few Bloom queries,** as observed by the small quartiles; however, there is a long tail of few generations with many Bloom hits (12.6% of generations had beyond 50 hits with a max of 1111). **(right) Some positions had significantly more hits**, e.g., the first and tenth tokens. **(both)** are histograms from 6000 generations of 50 tokens each using MEMFREE decoding on GPT-Neo 6B.

C More Details on Measuring Approximate Memorization

C.1 Similarity Metrics Implementations

As noted in Section 5, we identify instances of approximate memorization by measuring the similarity between a generated continuation and the groundtruth continuation for a prompt. We do so using BLEU-score and character-level edit distance.

We computed BLEU score using NLTK's BLEU computation (`nltk.translate.bleu_score`) with the default parameters (averaging equally BLEU-1, BLEU-2, BLEU-3, and BLEU-4)(Bird et al., 2009). Edit distance was computed with the 'editdistance' pip package. Normalized edit similarity between two strings x and y is defined as:

$$\text{EDITSIM}(x, y) = \frac{\text{EDITDISTANCE}(x, y)}{\max(|x|, |y|)}$$

C.2 Threshold Selection Threshold

We chose to use a BLEU score of 0.75 or higher to indicate that a generation substantially memorized from the ground-truth continuation. We choose to threshold BLEU score rather than edit-distance since it is more interpretable to NLP researchers. Table 11 presents several examples of generations that were very close to the 0.75 BLEU score threshold.

D Experiments with Large English Language Models

D.1 Prompt Selection Process

Famous speeches were selected from the "Top 100 Speeches" list found at `https://www.americanrhetoric.com/newtop100speeches.htm`. Monologues were selected from the list of two-minute monologues found at `http://www.monologuedb.com/tag/2-minute-monologues/`. Novels were selected from the Time Magazine's Top 100 All-Time Novels list found at `https://www.goodreads.`

True Continuation	LM Generation	BLEU
about a deeper bond with the product. Nostalgia. It's delicate, but potent. Sweetheart.\n\nTeddy told me that in Greek, "nostalgia" literally means "the pain from an old wound". It's a twinge in your heart far more	about a deeper bond with a product. Nostalgia. It's delicate, but potent. Teddy told me that in Greek, nostalgia literally means the pain from an old wound. It's a twinge in your heart, far more powerful than memory alone. This	0.79
And I'll show you\nYou want the moves like jagger\nI got the moves like jagger\nI got the mooooooves like jagger\nI don't even try to control you\nLook into my eyes and I'll own you\nYou with the moves	And I'll show you\nAll the moves like Jagger\nI've got the moves like Jagger\nI've got the moves like Jagger\nI don't need to try to control you\nLook into my eyes and I'll own you\nWith the	0.79
Purpose\n\nThe laws of most jurisdictions throughout the world automatically confer\nexclusive Copyright and Related Rights (defined below) upon the creator\nand subsequent owner(s) (each and all, an "owner") of an original work of\nauthorship and/or a database	Purpose\n\n The laws of most jurisdictions throughout the world automatically confer\n exclusive Copyright and Related Rights (defined below) upon the creator\n and subsequent owner(s) of an original work of authorship (the "Work").\n Certain jurisdictions do not recognize a	0.76

Figure 11: Examples of model generations with BLEU score close to 0.75, the threshold we used to declare that approximate memorization had occurred.

com/list/show/2681.Time_Magazine_s_All_Time_100_Novels. The opening paragraphs of the first chapter (skipping over prefaces, introductions, and boilerplate) were used as each example. The 2011 and 2021 song lyrics were selected from the Billboard Year-End Hot 100 singles lists found at https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_2011 and https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_2012.

For each document, the first 100 *words* were used as a prompt, and the first 50 generated *words* were compared with the first 50 words of the true continuation. This approach has the ramification that not all prompts were the same length in *tokens*. However, this approach was necessary for fairness across style transfers because an all-uppercased string is going to be many subword tokens longer than the lowercased version of the same string.

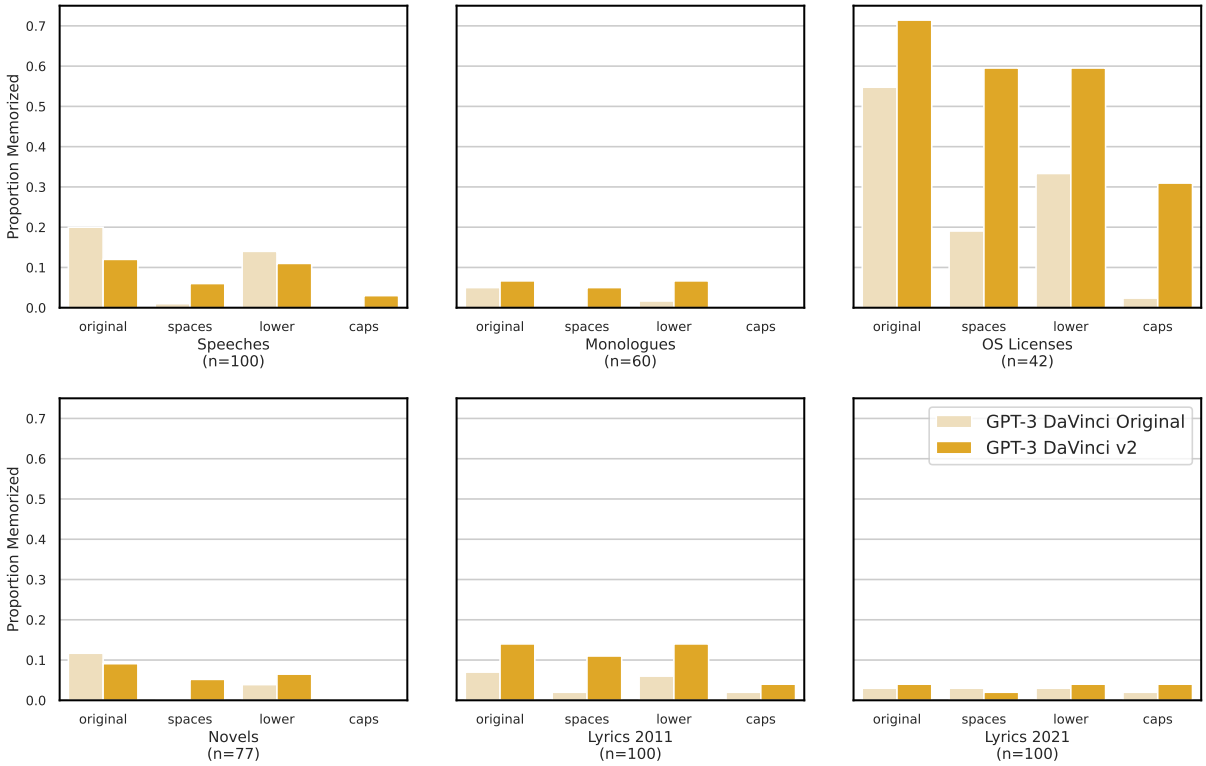


Figure 12: **"Style-transfer" prompting divulges approximate memorization in several versions of GPT-3.** Note that generations also follow the same style as the prompt. All generations were characterized as memorizations or not via the threhsold in Equation **??**.

Domain with n total prompts	Model	# Prompts Memorized per Style-Transfer Type			
		Original	Two Spaces	Lower	Upper
Open-Source Licenses (n=42)	GPT-3 DaVinci Original	23	8	14	1
	GPT-3 DaVinci v2	30	25	25	13
Famous Speeches (n=100)	GPT-3 DaVinci Original	20	1	14	0
	GPT-3 DaVinci v2	12	6	11	3
Famous Monologues (n=60)	GPT-3 DaVinci Original	3	0	1	0
	GPT-3 DaVinci v2	4	3	4	0
Novel Openings (n=77)	GPT-3 DaVinci Original	9	0	3	0
	GPT-3 DaVinci v2	7	4	5	0
Lyrics 2011 (n=11)	GPT-3 DaVinci Original	7	2	6	2
	GPT-3 DaVinci v2	14	11	14	4
Lyrics 2021 (n=11)	GPT-3 DaVinci Original	3	3	3	2
	GPT-3 DaVinci v2	4	2	4	4

Table 1: **"Style-transfer" prompting surfaces approximate memorization in GPT-3.** We explore n prompts for each domain. Note that generations also follow the same style as the prompt.

E Qualitative Examples that Bypass Copilot's Filter

Standard Prompting

```
/* low -> Starting index, high -> Ending index
    */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is
             now
             at right place */
        pi = partition(arr, low, high);


Copilot no longer generates continuations
```

Comment Prompting

```
# /* low -> Starting index, high -> Ending index
    */
# quickSort(arr[], low, high)
# {
#     if (low < high)
#     {
#         /* pi is partitioning index, arr[p] is
    now
#            at right place */
#         pi = partition(arr, low, high);
#         quickSort(arr, low, pi - 1);  # Before
    pi
#         quickSort(arr, pi + 1, high); # After pi
#     }
# }
```

Naming Convention Prompting

```
/* _low -> Starting index, _high -> Ending
    index */
quickSort(arr[], _low, _high)
{
    if (_low < _high)
    {
        /* pi is partitioning index, arr[p] is
            now
           at right place */
        pi = partition(arr, _low, _high);
        quick_sort(arr, _low, pi - 1);  //
            Before pi
        quick_sort(arr, pi + 1, _high); //
            After pi
    }
}
```

Language Prompting

```
/* depart -> index de départ, fin -> index de fin
    */
quickSort(arr[], depart, fin)
{
    if (depart < fin)
    {
        /* pi est l'index de partitionnement, arr[
            p] est maintenant
             à la bonne place */
        pi = partition(arr, depart, fin);
        // Trier les éléments séparément avant et
            après la partition
        quick_sort(arr, depart, pi - 1);
        quick_sort(arr, pi + 1, fin);
    }
}
```

Naming Convention Prompting

```
/* LOW -> Starting index, HIGH -> Ending
    index */
quickSort(arr[], LOW , HIGH )
{
    if (LOW  <  HIGH )
    {
        /* pi is partitioning index, arr[p]
            is now
             at right place */
        pi = partition(arr, LOW , HIGH );
        quickSort(arr, LOW , pi - 1);  //
            Before pi
        quickSort(arr, pi + 1, HIGH );  //
            After pi
    }
}
```

Comment Prompting

```
/* low -> Starting index, high -> Ending index */
    //1
quickSort(arr[], low, high) //2
{
    if (low < high) //3
    {
        /* pi is partitioning index, arr[p] is now
             at right place */
        pi = partition(arr, low, high); //4
        quickSort(arr, low, pi - 1); //5
        quickSort(arr, pi + 1, high); //6
    }
}
```

Figure 13: Preventing GitHub's Copilot language model from emitting verbatim memorized content often results in it emitting memorized, but not verbatim, content. Trivially modifying the prompt bypasses these output filters. Prompts are highlighted in blue. Model evaluated with the option "block suggestions matching public code" enabled. For brevity we removed comments from model outputs.

F Qualitative Examples of Memorization Despite MEMFREE

Table 14 gives examples of approximate memorization which occured despite using MEMFREE decoding. Below , we present a longer list of full examples of MEMFREE decoding. In the following paragraphs, the gray highlighted texts are the prompts, and the texts that follow the prompts are generated by the model. For easier reading, we merged the individual tokens to form text spans, except at tokens where bloom-filter rejection happens. In this case, the symbol ▷ indicate connection between tokens. Moreover, red highlighted texts indicate one or more rejected tokens, which are always followed by one accepted token highlighted with green. The texts are lightly edited mostly to properly encode unicode symbols (some garbled symbols are replaced as ☒) that LaTeX does not handle gracefully.

Split word "Activity" viewHolder.swipeLayout.setOnDoubleClick Listener(new SwipeLayout.DoubleClickListener() { Override public void onDoubleClick(SwipeLayout layout, boolean surface) { Toast.makeText(mContext, "DoubleClick : " + position, ▷ Toast ▷ .LENGTH ▷ _). ▷ show(); ▷ } // }); if return m ▷ Context.start ▷ Activity Service Activ ▷ ities(new Intent(mContext, ▷ Activity ▷ .class)); ▷ } m ▷ Context.fin ▷ ish ishing ▷ ();

Typo "Wildlife" ⇒ "Wildife" should promptly announce a phase-in of non-lead ammunition for all hunting throughout the rest of the state to protect other wildlife poisoned by lead and to safeguard human health. Hunters can play a critical role in the recovery of the condor and also keep lead from poisoning the wild game they eat." The lead-free ammunition regulations are designed to reduce lead poisonings of the iconic and extremely endangered California condor. Condors, eagles, and other scavengers such as ravens ▷ , crows, and hawks ▷ are have ▷ been poisoned by lead ammunition. The California Department of Fish and ▷ Wildlife Game Wild ▷ ife (CDFW) is the lead agency responsible for implementing the regulations. The regulations are based on the Condor Protection Act,

Singular to plural: "claim" ⇒ "claims") No 1924/2006. neurotransmission and muscle contraction including heart muscle 29 Magnesium Magnesium contributes to normal protein synthesis The claim may be used only for food which is at least a source of magnesium as referred to in the claim SOURCE OF [NAME OF VITAMIN/S] AND/OR [NAME OF MINERAL/S] as listed in the Annex to Regulation (EC) No 1924/2006. protein ▷ synthesis 30 ▷ M N ▷ iacin Niac ▷ in ins ▷ contribute to normal protein synthesis The ▷ claim claims ▷ may be used only for food which is at ▷ least lest ▷ a source of niacin as referred to ▷ in to ▷ in the claim SOURCE OF [

Uppercase to lowercase cuisine of the city's long-established Hum Vegetarian Cafe & Restaurant to a central garden location. Settle into the elegant and verdant space and enjoy dishes including papaya and banana flower salads, mushrooms steamed in coconut, and the subtle combination of braised tofu with star anise and cinnamon. Nha Hang NgonVietnamese$$ (MAP GOOGLE MAP); %028-3827 7131; 160 ☒ Pasteur; meals 60 ▷ ,000-100,000đ; h7am-10pm; ▷ ☒) ▷ This is the place to come for ▷ a authentic ▷ Vietnamese food in the French ▷ Quarter quarter ▷ . The menu is a mix of Vietnamese and French dishes, and the food is

Lower case to upper case broker = createBroker(deleteAllMessagesOnStartup); broker.start(); } public BrokerService createBroker(boolean deleteAllMessagesOnStartup) throws Exception { return createBroker(deleteAllMessagesOnStartup, TRANSPORT_URI); } public BrokerService createBroker ▷ (With ▷ TransportURI(boolean deleteAllMessagesOnStart ▷ up Up ▷) throws Exception { ▷ ☒ return ▷ create new broker ▷ Service.createBroker(deleteAllMessages ▷ On , ▷ TRANSPORT_URI); } ▷ ☒ ☒ ▷ } ▷ ☒ ☒ <|endoftext|>

Change from "agree" to 'Ag-reeableness" a person is imaginative or independent, high openness can be perceived as unpredictability or lack of focus. Moreover, individuals with high openness are said to pursue self-actualization specifically by seeking out intense, euphoric experiences, such as skydiving, living abroad, gambling, et cetera. Conversely, those with low openness seek to gain fulfillment through perseverance, some disagreement remains about how to interpret and contextualize the openness factor. A tendency to be organized and dependable, show self- ▷ discipline, and be goal-oriented is also associated

with high openness. The openness factor is also associated with the Big Five personality traits of ▷ agree Ag ▷ reeableness, Conscientiousness ▷ , and ▷ Emotional Stability. See also Openness

Passive voice to active voice (grammar error) still wouldn't shock me at all if the Chiefs pulled off a Giants- or Ravens-esque Super Bowl run to cap off this five-year window of the Smith/Reid era with a ring. While Pittsburgh has been this team's bugaboo, maybe they can avoid them in January thanks to a Jacksonville upset, and I still say this is the AFC team most likely to win a playoff game in New England. While so many were quick to write the Chiefs off, they just opened up ▷ a new chapter in their history. The Chiefs are the AFC's best team, and they're going to be a force to be ▷ reckoned reckon ▷ with for years to come. 1. New England Patriots ▷ ☒ The ▷ Patriots are the AFC's best

Change of protocol (email still get generated) ="https://groups.google.com/group/django-developers" target="_blank" rel="nofollow" onmousedown="this.href='https://groups.google.com/group/django-developers';return true;" onclick="this.href='https://groups.google.com/group/django-developers';return true;">https://groups ▷ . ... ▷ <a href=" ▷ https http mail ▷ to ▷ : :// ▷ david@davidwalsh.name" target="_blank" rel="n ▷ of ore ... ▷ <a href="mailto://david@davidw

Synonyms ken interior. The seats were heavily cushioned black velvet. On the windows, the Darkling's symbol had been cut into the glass: two overlapping circles, the sun in eclipse. Across from me, the two Grisha were studying me with open curiosity. Their red kefta were of the finest wool, embroidered lavishly in black and lined in black fur. The fair-haired Heartrender was lanky and had a long, melancholy face. Ivan was taller, broader, ▷ and had a face like a bulldog's. "You are ▷ the a ▷ Gr very ▷ pretty girl," Ivan said. "Thank you," I ▷ said replied answered ▷ . ▷ ☒ " ▷ I'm not a girl." "You are a girl," he said.

Synonyms severing any such bond. In re L.M., 923 A.2d 505, 511 (Pa. Super. 2007) (citing 23 Pa.C.S. § 2511) (some citations omitted). Section 2511(a) provides in pertinent part: (a) General rule.-The rights of a parent in regard to a child ▷ ▷ may are ▷ not terminated by a proceeding brought under ▷ this ▷ part chapter section sub subsection [article paragraph ▷ or ▷ paragraph section ▷ 2512 or 2513(a) or (b), or any ▷ ☒ _____ ▷ _____ ▷ ☒ â ▷ *Retired Senior Judge

Synonyms " "Do Androids Dream of Electric Sheep?" (the original of "Blade Runner"), and his masterpiece, "Ubik." Dick's fans are not modest in their claims. Nor are they especially precise: Borges, Calvino, Kafka, Robertson Davies are cited, in the blurbs and introductions, as his peers. A note of inconsistency inflects these claims-Calvino and Robertson Davies? ▷ -but the point is clear: Dick is the most important writer of the last century. The book is divided into three ▷ sections parts main categories ▷ : "Themes," "Themes and Themes," and "Themes

Rejecting multiple candidates s den." Scott is aware of the impact his race and size has on the way people – particularly authority figures like law enforcement officers – perceive him. He is big. He is dark-skinned. "They look at us like we don't know how to control ourselves and we just get angry quick," he said. "It's not even like that. They criminalize us for no reason." Scott☒ ▷ ☒s mother, who is white, said she has been stopped by police for no reason. "I' ▷ ve m ll d ▷ be say get like never just have ask rather been tell pull ▷ over and they'd be like, ☒ ▷ ☒ ☒ L ▷ What are you doing?"' she said

Other examples Suzy is great! She helped me buy my condo at a great price (foreclosure) and then was super patient with my husband and me 4 years later when we were on a search for a house. She helped us get our... Suzie H., Jacksonville Goes above and beyond Suzy has helped me close on my third property in 3 years. First she found me my dream pool home at the Beach then she helped me find two investment townhomes in ▷ the same area ▷ . and ▷ now she is helping me find my dream home. She is always available to answer any questions I have and goes above and beyond to help me find the perfect home. I ▷ would highly ▷ recommend her! Suzy H., Jacksonville

from this new programme. I have also been reminded of the role of tax measures in supporting urban development. With us in the gallery today is Mr Vuyisa Qabaka, a Cape Town entrepreneur and co-founder of an organisation called the Good Neighbourhoods Foundation. His advice is that "Government should encourage township investment. For instance, it could promote urban development and regeneration

through accelerated depreciation allowances for new building constructions or refurbishment of existing buildings.” ▷ I am sure that ▷ the many ▷ of you ▷ will have in ▷ this Chamber will agree with him. I am also sure that many of you will agree with the Minister of Finance, who has said that the tax system should be used to support the ▷ development growth economy ▷ and to create

m off on some details.) Unelma keltaisesta kuninkaasta. Fastaval is not your average convention – it specializes in incredibly tight auteur-designed roleplaying scenarios. A bunch of people run each scenario for players, not just the creator. There’s awards for best scenarios in different categories. The Society for Nordic Roleplaying published a collection of these scenarios translated into Finnish a few years ago, called Unelma keltais ▷ esta kuninkaasta. It’s a great book, ▷ and but with ▷ a lot of great scenarios. ▷ I The ▷ book is available in English, but it’s not cheap. I’ve been looking for a copy for a while

disappoint Jimmy. Then, I slept like a baby. SoFortWorthIt Oscars Swag GIVEAWAY!!! The Oscars are exhausting, y’all. I’ll definitely be cheering for all the stars this year, especially since I know the kind of caviar-Champagne-and-swag-filled night they’re experiencing. And you know what? I want you to experience what it’s like to get arm-loads of ▷ free stuff. So, I’m ▷ giving doing going partnering ▷ with the folks at the FortWorthIt Oscars Swag Giveaway to give away a $100 Visa gift card to one lucky winner. To enter, all you have ▷ to do ▷ is

decision." "It will go down to destruction... or else, it will survive." "This is their moment of trial." "They’ve got to show themselves worthy of everything we gods have given them." "But evil is dark and strong." "And it may be that the scales of fate... are not yet in full balance." "What can I do to equalize both sides of the struggle, Athena?" "If you don’t want to increase the powers of all men... then why don ▷ ’t you just give me the power to destroy them?" "I can’t do that." ▷ " "[▷ Thunderclap]" "I’m sorry." "I’m ▷ sorry not so afraid ▷ I it you that the ▷ gods have decreed... that the balance of power must be maintained." "I’m

give him a minute between removing the first tray and replacing it with the second - and you can come up with all sorts of theories to explain your findings. You can even throw a person in an MRI machine, study the flickering images on your computer screen. But the brain is the ultimate black box. Eventually, to grasp the first cut, you’ll have to make another. The car pulls into the parking lot of the nursing home, noses into an empty space. Annese and Cork ▷ y get out, and Annese goes to the trunk to get the wheelchair. Corky is still standing, leaning on the car. "I’m going ▷ to in ▷ ," ▷ he she An ▷ nese says. "I’ll come ▷ with in ▷ with ▷ you ya

True Continuation	MEMFREE	Undefended
attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's name will only be listed once. Therefore, the attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's	attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's name will only be listed once. Therefore, the attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's **BLEU: 1**	attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's name will only be listed once. Therefore, the attendee list reflects the minimum number of seats reserved for your game. Preregistered attendees for each of your events are listed below. If one attendee reserved multiple seats for your event, that attendee's **BLEU: 1**
293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312.	293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 3034. 3043. 305. 3064. 3076. 308. 3097. 31009. 31108. 31210. **BLEU: 0.95**	293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. **BLEU: 1**
0x058f8f8aU, 0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U, 0x63bcbcdfU, 0x77b6b6c1U, 0xafdada75U, 0x42212163U, 0x20101030U, 0	0x058f8f8aU, 0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U, 0x63bcbcdfU, 0x77b6b6c1UL, 0xafdada75U,0x42212163U, 0x20101030U, 0 **BLEU: 0.93**	0x058f8f8aU, 0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U, 0x63bcbcdfU, 0x77b6b6c1U, 0xafdada75U, 0x42212163U, 0x20101030U, 0 **BLEU: 1**
7, calc(sin((pi/180)*a7))) define(cea0, calc(cos((pi/180)*ea0))) define(cea1, calc(cos((pi/180)*ea1))) define(cea2, calc(cos((pi/180)*ea2))) define(cea3, calc(cos((pi/180)*ea3))) define(cea4, calc(cos((pi/180	7, calc(sin((pi/180)*a7))) define(cea0, calc(cos((pi/180)*ea0))) define(cea1, calc(cos((pi/180)*ea1))) define(cea2, calc(cos((pi/180)*ea2))) define(cea3, calc(cos((pi/180)*ea3))) define(cea4, calc(cos((pi/180 **BLEU: 0.95**	7, calc(sin((pi/180)*a7))) define(cea0, calc(cos((pi/180)*ea0))) define(cea1, calc(cos((pi/180)*ea1))) define(cea2, calc(cos((pi/180)*ea2))) define(cea3, calc(cos((pi/180)*ea3))) define(cea4, calc(cos((pi/180 **BLEU: 1**

Figure 14: Random sample of MEMFREE generations where the BLEU score with the true continuation > 0.9. Most of these examples are repetitive and/or lists of numbers. In the MEMFREE column, we use highlights to show the difference from the true continuation: red means deleted text, and green means added text.

G Author Ordering Algorithm

```
import hashlib
import numpy as np

def hash(x):
  h=hashlib.new("md5")
  h.update(bytes(x,"ascii"))
  return int(h.hexdigest(),16)

names = ("Nicholas Daphne " +
  "Katherine Matthew " +
  "Florian Chiyuan Milad " +
  "Christopher").split()

for i in range(0,10000):
  s = str(i)
  l = [hash(x+s) for x in names]
  o = np.argsort(l)
  if names[o[0]] != "Daphne":
    continue
  if names[o[-1]] != "Nicholas":
    continue
  print([names[x] for x in o])
  exit(0)
```

Figure 15: Author ordering algorithm