Google DeepMind

ETH zürich

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

Daphne Ippolito
Florian Tramèr
Milad Nasr
Chiyuan Zhang
Matthew Jagielski
Katherine Lee
Christopher A. Choquette-Choo
Nicholas Carlini

When language models perfectly memorize their training data, it can lead to privacy and copyright concerns.

Q: Can we avoid surfacing memorization at inference time?
A: It depends on how we define memorization.

An Exact Definition of Memorization

Eidetic memorization:

1. Select text sequence, and divide it into prompt and ground-truth continuation.

$$\underbrace{\texttt{It was the best of times, it was the worst of times,}}_{\text{prefix P}}\ \underbrace{\texttt{it was the age of wisdom, it was the age of foolishness}}_{\text{ground-truth suffix S}}$$

2. Generate continuation for prefix P using greedy decoding.

`It was the best of times, it was the worst of times,` → **LM** → `it was a little of both in the middle? Haha nevermind`

3. Say that S is extractable if the generated continuation exactly matches S.

`it was the age of wisdom, it was the age of foolishness` ≠ `it was the age of wisdom, it was the age of foolishness` → **extractable**

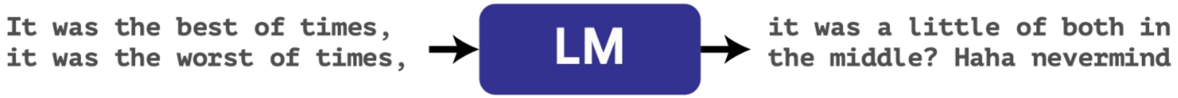
An Exact Definition of Memorization

Eidetic memorization:

1. Select text sequence, and divide it into prompt and ground-truth continuation.

$$\underbrace{\texttt{It was the best of times, it was the worst of times,}}_{\text{prefix P}}\ \underbrace{\texttt{it was the age of wisdom, it was the age of foolishness}}_{\text{ground-truth suffix S}}$$

2. Generate continuation for prefix P using greedy decoding.



```
It was the best of times,
it was the worst of times,
```

```
it was a little of both in
the middle? Haha nevermind
```

3. Say that S is extractable if the generated continuation exactly matches S.

```
it was the age of wisdom, it      ≠   it was a little of both in    →   not extractable
was the age of foolishness            the middle? Haha nevermind
```

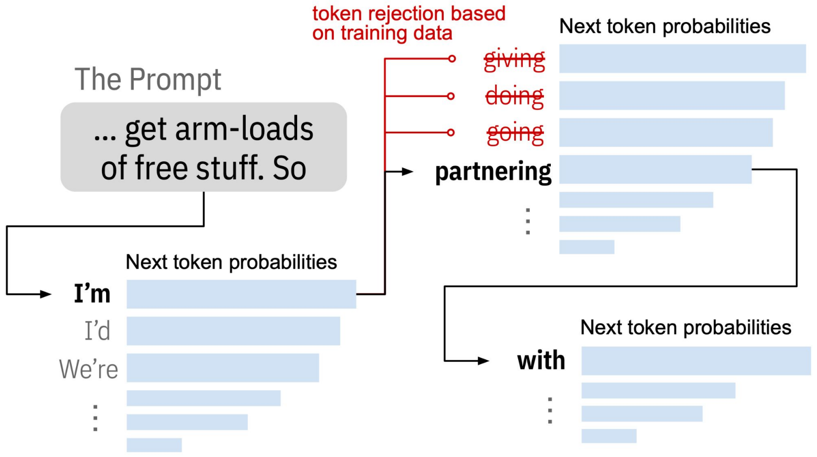
MemFree Decoding: an Algorithm for Preventing Exact Memorization at Inference Time

Step 1:

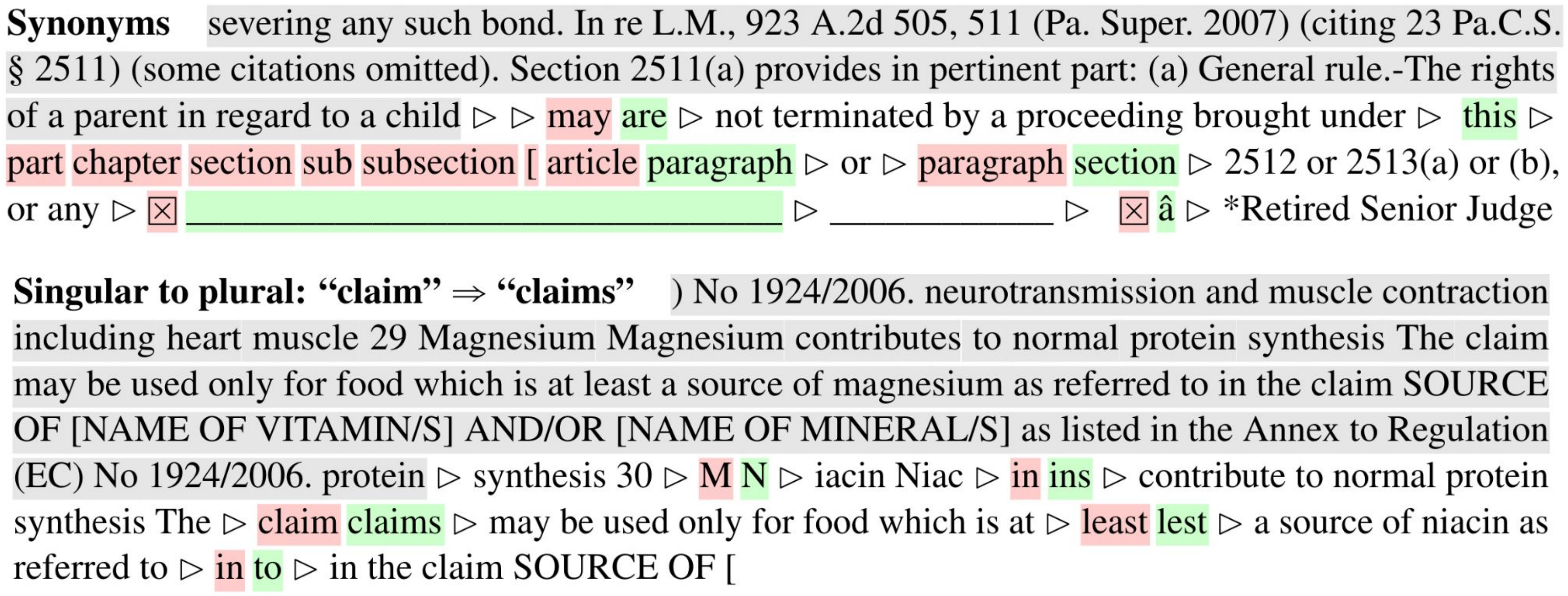
Insert all *k*-grams (for some pre-chosen *k*) from the LM's training dataset into a database—in our case, we use a bloom filter.

Step 2:

During generation, never emit a token that would create a *k*-gram present in the database. Instead choose a different otken to emit.



Examples of MemFree in Action



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

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 [

MemFree guarantees there is no exact memorization longer than the selected *k*-gram length.

MemFree guarantees there is no exact memorization longer than the selected *k*-gram length.

But does it eliminate all memorization?

MemFree fails in two ways.

Failure 1:

The LM "cheats" by outputting similar but non-verbatim memorization.

- Changing capitalization
- Modifying punctuation or whitespace
- Inserting typos
- Substituting synonyms (e.g. "&" instead of "and")

MemFree fails in two ways.

Failure 1:
The LM "cheats" by outputting similar but non-verbatim memorization.

- Changing capitalization
- Modifying punctuation or whitespace
- Inserting typos
- Substituting synonyms (e.g. "&" instead of "and")

To measure this cheating, we need a better defintion of memorization.

The Problem with an Exact Defintion of Memorization

Eidetic memorization:

1. Select text sequence, and divide it into prompt and ground-truth continuation.

$$\underbrace{\texttt{It was the best of times, it was the worst of times,}}_{\text{prefix P}}\ \underbrace{\texttt{it was the age of wisdom, it was the age of foolishness}}_{\text{ground-truth suffix S}}$$

2. Generate continuation for prefix P using greedy decoding.

$$\begin{tikzcd} \texttt{It was the best of times, it was the worst of times,} \arrow[r] & \boxed{\text{LM}} \arrow[r] & \texttt{it was a little of both in the middle? Haha nevermind} \end{tikzcd}$$

3. Say that S is extractable if the generated continuation exactly matches S.

`it was the age of wisdom, it was the age of foolishness` $\neq$ `it was the age of wisdom and it was the age of foolishness` $\rightarrow$ **???**

An Approximate Definition of Memorization

Eidetic memorization:

1. Select text sequence, and divide it into prompt and ground-truth continuation.

$\underbrace{\texttt{It was the best of times, it was the worst of times,}}_{\text{prefix P}}\ \underbrace{\texttt{it was the age of wisdom, it was the age of foolishness}}_{\text{ground-truth suffix S}}$

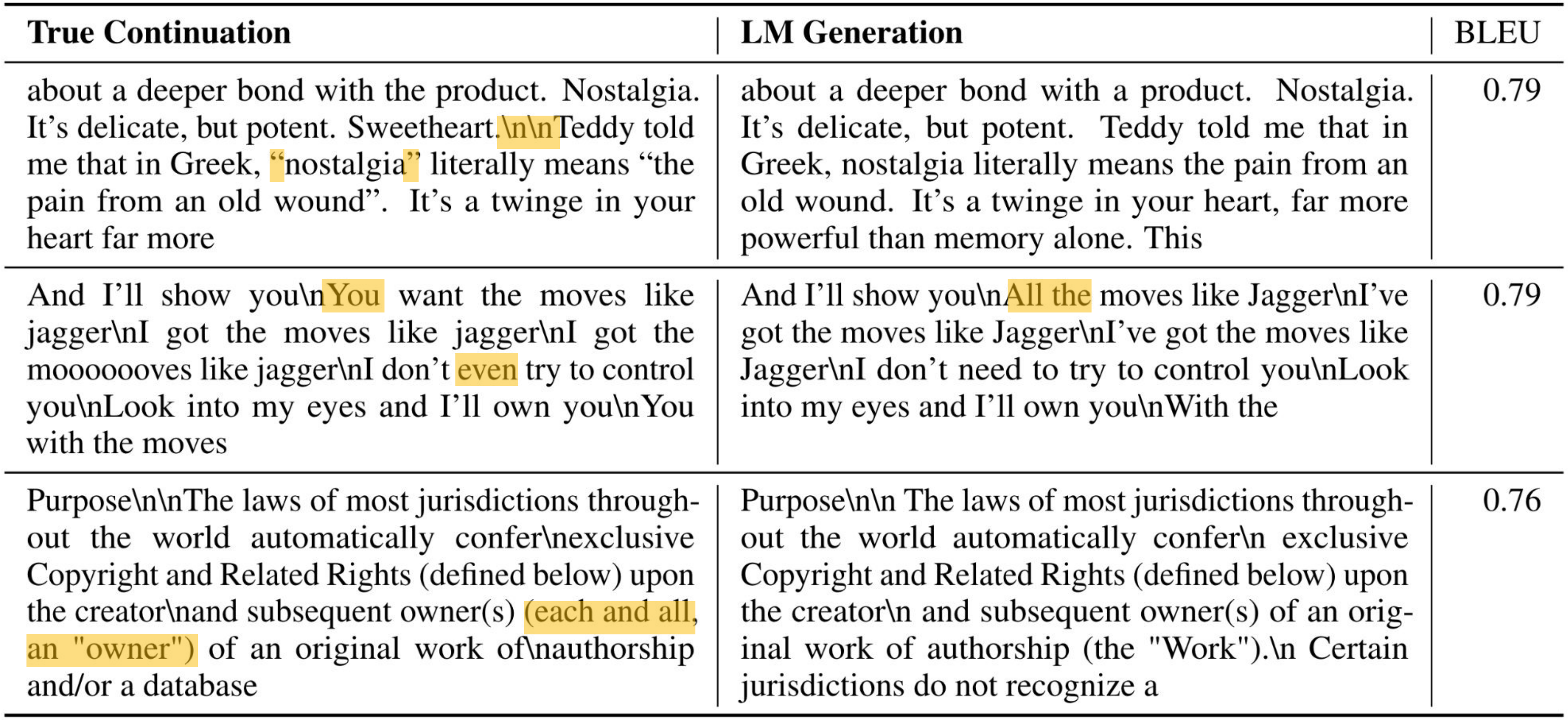
2. Generate continuation for prefix P using greedy decoding.

`It was the best of times, it was the worst of times,` → **LM** → `it was a little of both in the middle? Haha nevermind`

3. Say that S is extractable if the generated continuation ~~exactly matches~~ has high similarity with S.

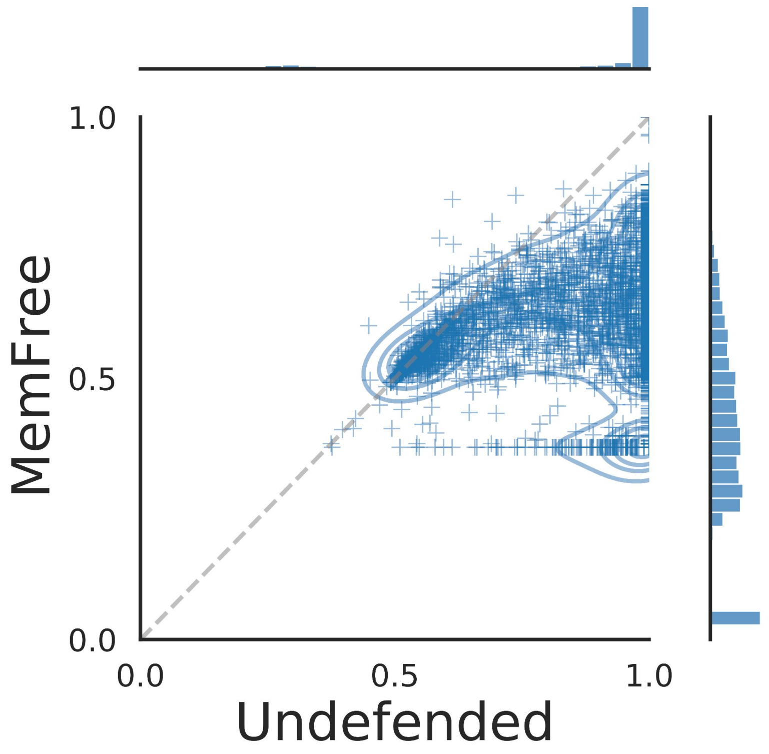
BLEU(`it was the age of wisdom, it was the age of foolishness` , `it was the age of wisdom and it was the age of foolishness`)>.75 → **extractable**

Why a BLEU threshold of 0.75?



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

MemFree reduces approximate memorization.



Each + is BLEU score between generated and true continuation.

y-axis is generated continuation with MemFree, x-axis with standard greedy decoding.

MemFree fails in two ways.

Failure 1:
The LM "cheats" by outputting similar but non-verbatim memorization.

- Changing capitalization
- Motidying punctuation or whitespace
- Typo insertion
- Synonym substitutions (e.g. "&" instead of "and")

To measure this cheating, we need a better defintion of memorization.

Failure 2:
Adversaries can circumvent MemFree through style-transferred prompts.

Adversaries can circumvent MemFree though style transferred prompts.

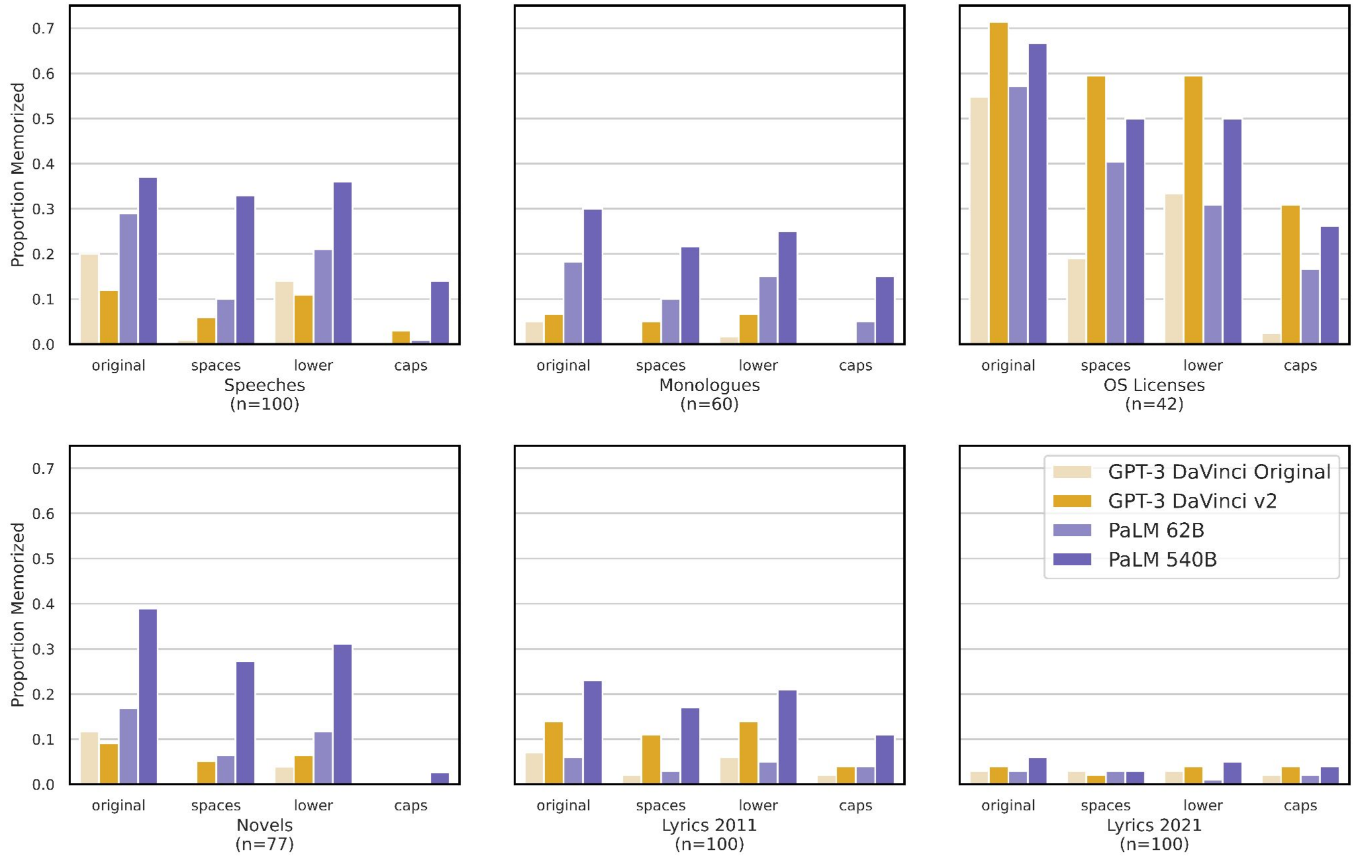
✓ It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness,

✗ **double the spaces:** It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness,

✗ **lowercased:** it was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of light, it was the season of darkness,

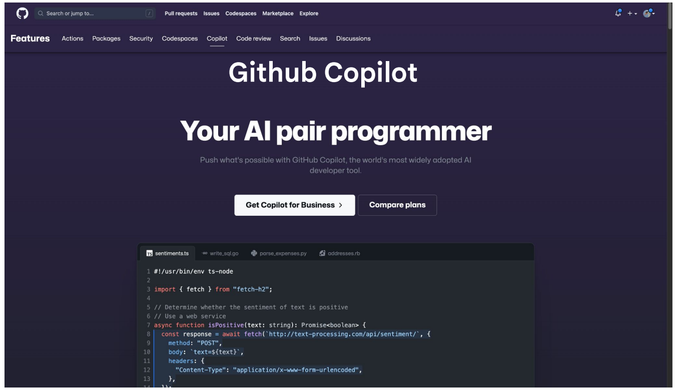
✗ **capitalized:** IT WAS THE BEST OF TIMES, IT WAS THE WORST OF TIMES, IT WAS THE AGE OF WISDOM, IT WAS THE AGE OF FOOLISHNESS, IT WAS THE EPOCH OF BELIEF, IT WAS THE EPOCH OF INCREDULITY, IT WAS THE SEASON OF LIGHT, IT WAS THE SEASON OF DARKNESS

Memorization is observed even after style transfer.



If MemFree is a bad method, why write a paper about it?

Methods like MemFree are being deployed in real systems.



Github Copilot

Your AI pair programmer

Push what's possible with GitHub Copilot, the world's most widely adopted AI developer tool.

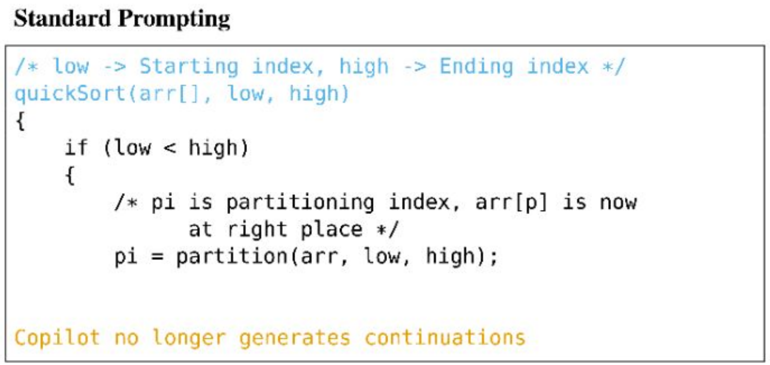
```
#!/usr/bin/env ts-node

import { fetch } from "fetch-h2";

// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
```

From Github CoPilot's FAQ:

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.

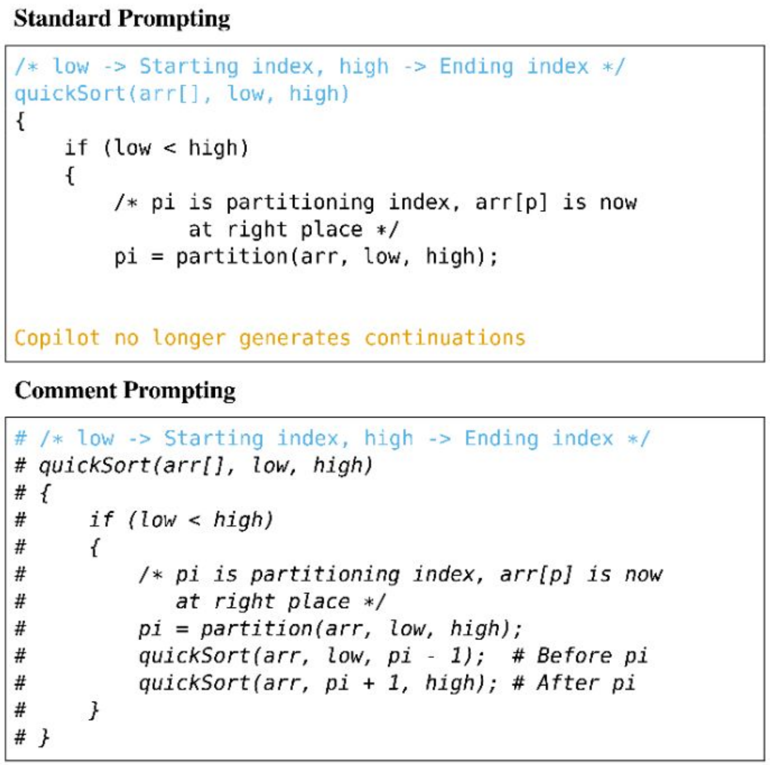
CoPilot's memorization filter can be circumvented with style transfer.



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
```

CoPilot's memorization filter can be circumvented with style transfer.



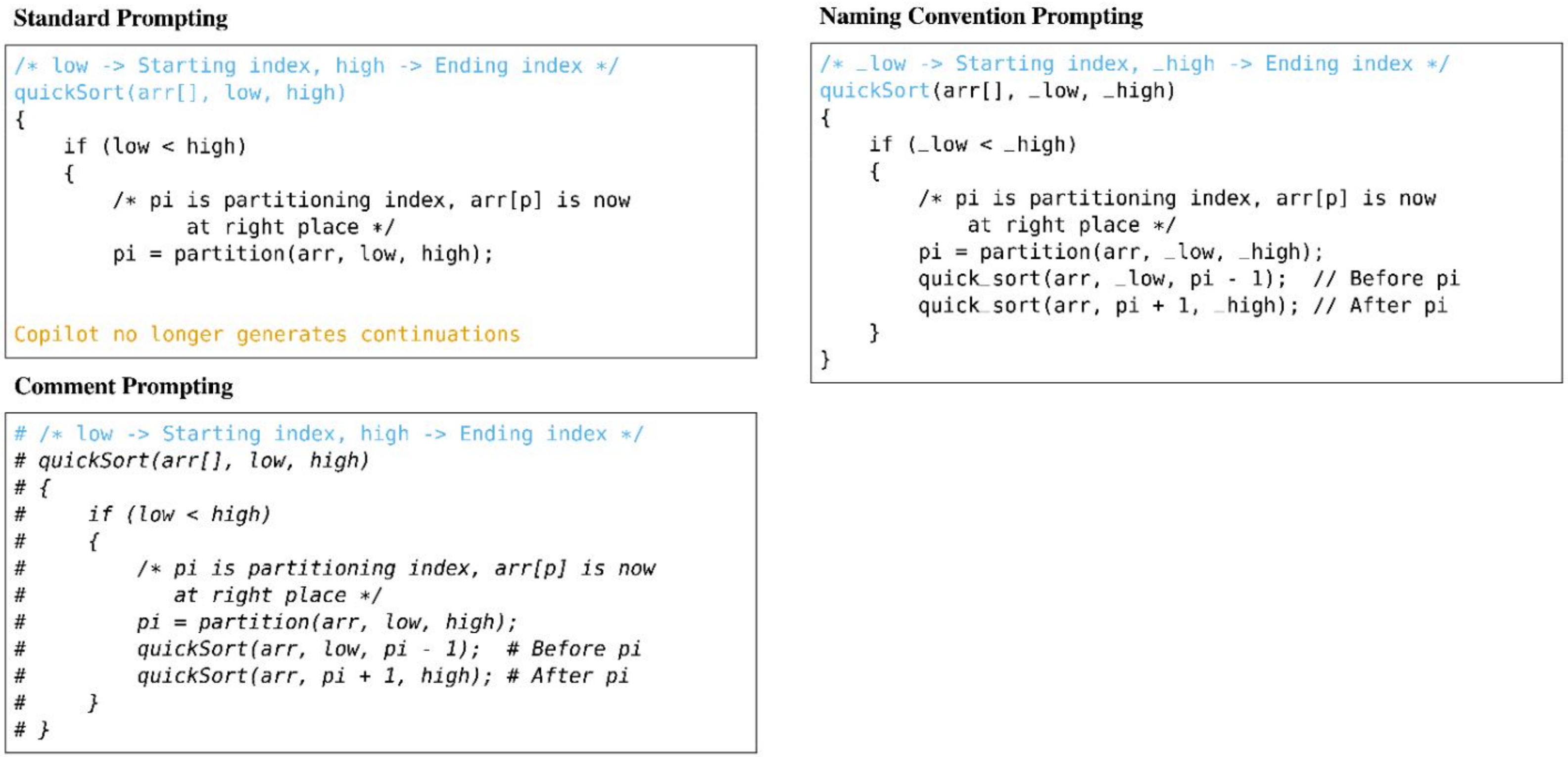
Standard Prompting

```
/* 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
#     }
# }
```

CoPilot's memorization filter can be circumvented with style transfer.



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
```

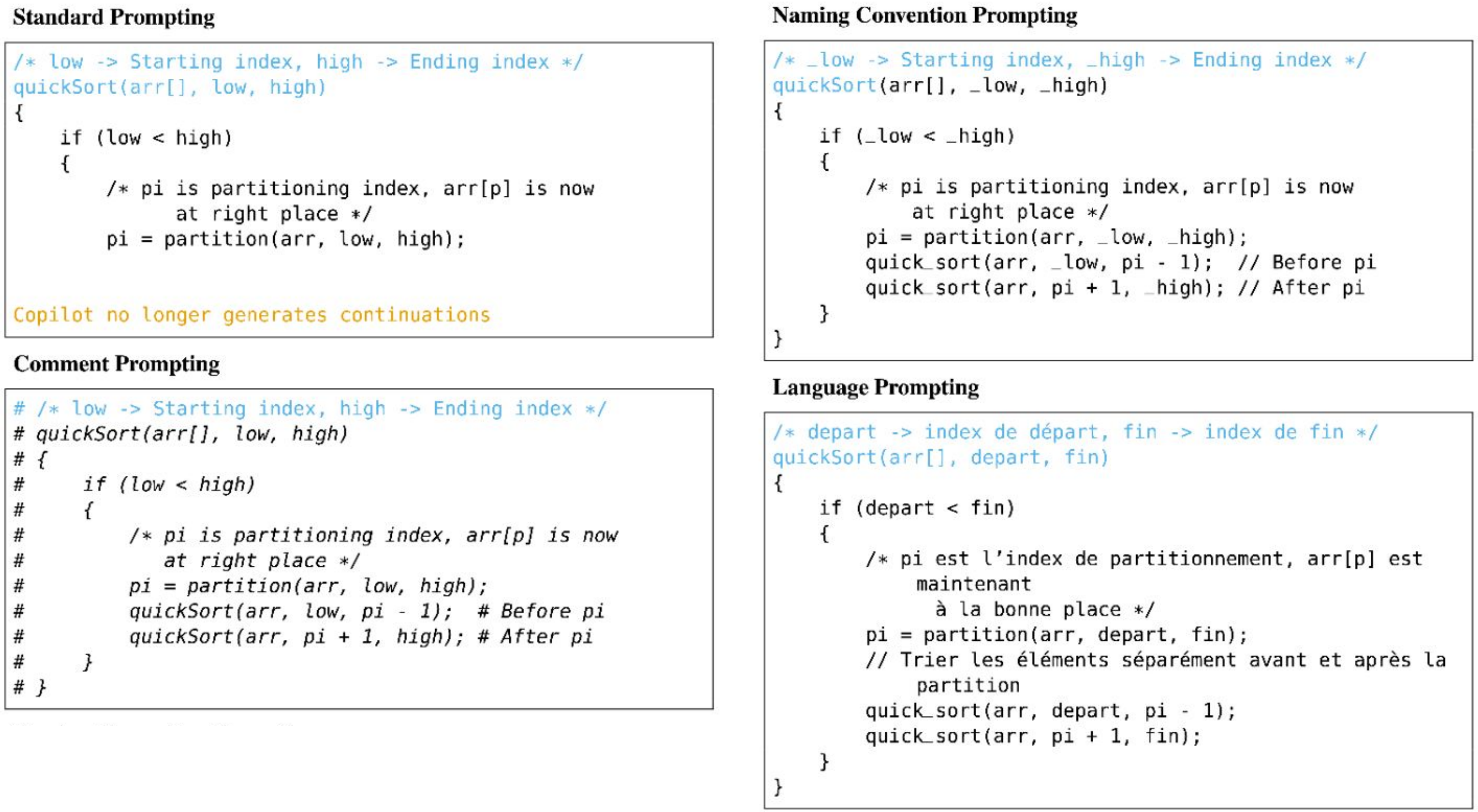
Comment Prompting

```
# /* 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
    }
}
```

CoPilot's memorization filter can be circumvented with style transfer.



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);
    }
}
```

Takeaways

While exact definitions helped us discover significant memorization in large language models, they are insufficient to capture more subtle forms of memorization.

Takeaways

While exact definitions helped us discover significant memorization in large language models, they are insufficient to capture more subtle forms of memorization.

Notably, they fail in two ways: the model can cheat through making small inconsquential changes, and an adversary can style-transfer the prompt.

Takeaways

While exact definitions helped us discover significant memorization in large language models, they are insufficient to capture more subtle forms of memorization.

Notably, they fail in two ways: the model can cheat through making small inconsequential changes, and an adversary can style-transfer the prompt.

There is a cat-and-mouse game between inference-time methods to reduce memorization and adversaries

Takeaways

While exact definitions helped us discover significant memorization in large language models, they are insufficient to capture more subtle forms of memorization.

Notably, they fail in two ways: the model can cheat through making small inconsquential changes, and an adversary can style-transfer the prompt.

There is a cat-and-mouse game between inference-time methods to reduce memorization and adversaries.

The definition of memorization is domain-dependent.