Beyond Memorization: The Challenge of Random Memory Access in Language Models

Tongyao Zhu\textsuperscript{1,2} Qian Liu\textsuperscript{1} Liang Pang\textsuperscript{3} Zhengbao Jiang\textsuperscript{4} Min-Yen Kan\textsuperscript{2} Min Lin\textsuperscript{1}

\textsuperscript{1}Sea AI Lab  \textsuperscript{2}National University of Singapore  \textsuperscript{3}Institute of Computing Technology, CAS  \textsuperscript{4}Carnegie Mellon University
tongyang.zhu@u.nus.edu \{liuqian,linmin\}@sea.com
pangliang@ict.ac.cn zhengbaj@cs.cmu.edu knmyn@nus.edu.sg

Abstract

Recent developments in Language Models (LMs) have shown their effectiveness in NLP tasks, particularly in knowledge-intensive tasks. However, the mechanisms underlying knowledge storage and memory access within their parameters remain elusive. In this paper, we investigate whether a generative LM (e.g., GPT-2) is able to access its memory sequentially or randomly. Through carefully-designed synthetic tasks, covering the scenarios of full recitation, selective recitation and grounded question answering, we reveal that LMs manage to sequentially access their memory while encountering challenges in randomly accessing memorized content. We find that techniques including recitation and permutation improve the random memory access capability of LMs. Furthermore, by applying this intervention to realistic scenarios of open-domain question answering, we validate that enhancing random access by recitation leads to notable improvements in question answering. The code to reproduce our experiments can be found at https://github.com/sail-sg/lm-random-memory-access.

1 Introduction

Language models (LMs) have recently showcased outstanding abilities in NLP tasks with a large amount of memory stored in their parameters (Brown et al., 2020; Ouyang et al., 2022). Through pre-training on large text corpora, LMs memorize factual knowledge about the world (Zhou et al., 2023). Consequently, they show great performance in knowledge-intensive tasks (Petroni et al., 2021) such as open-domain question answering (Kamaloo et al., 2023; Ziemis et al., 2023; Mallen et al., 2023). There is a growing interest in considering LMs as knowledge bases (Wang et al., 2021; Heinzerling and Inui, 2021; Petroni et al., 2019; Cao et al., 2021; AlKhamsi et al., 2022). Despite the recent advances in applying LMs to solve downstream tasks, the fundamentals of how LMs store knowledge and access memory in their parameters remains a subject of ongoing research and intrigue (Tirumala et al., 2022; Zhu and Li, 2023; Allen-Zhu and Li, 2023; Berglund et al., 2023).

In this paper, we draw inspiration from memory-accessing patterns observed in computer systems to explore whether LMs can access their parametric memory in a sequential or random manner. We extrapolate these concepts to investigate LMs and delineate two memory access patterns: \textit{sequential memory access} means that the model starts from the beginning of a memorized sequence, progressing through the content in consecutive order. Conversely, \textit{random memory access} denotes that the model can commence from any location within the memorized content, without needing to start from the beginning. For instance, reciting a memorized poem line by line is considered sequential access, while directly starting from the third line involves random access.

With these concepts, we design experiments with both synthetic and real data to evaluate the language model’s ability to perform sequential or random access to memorized content, as illustrated in Figure 1. We limit our study to decoder-only language models due to their increasing popularity.

Figure 1: A illustration of our investigation of memory access pattern in language models. We find that the model accesses its parametric memory largely in a sequential manner, and faces difficulty in randomly accessing the content in the middle of memorized strings.
and capability (Radford et al., 2019; Brown et al., 2020; Touvron et al., 2023a,b; Jiang et al., 2023). We first ask the model to memorize key–value pairs of various types and show that the model is able to sequentially read memorized content to a satisfying degree. Next, we test the model’s random access ability by training it to recite a sentence or find an answer to a question in a memorized passage. In such tasks, the model’s performance falls drastically when it is required to extract a span in the middle of a passage, revealing its incapability to randomly access its memory.

Given that language models struggle to perform random access to their memory, we pursue two means for mitigation: recitation at inference time, and permutation during training. Recitation enables the model to sequentially read its parametric memory first before performing a task. The model’s performance can thus be enhanced by utilizing the recited content in its context window. We also show that simply permuting sentences in a passage during training to memorize content also improves performance.

We finally verify the challenge of random access through a case study on open-domain question answering. We reduce the difficulty of the task by allowing the model to memorize passages with ground-truth answers, yet we find that the model benefits the most from such memorization when it is allowed to recite a relevant passage and then answer the question. Overall, we make several contributions to further understand the memory access mechanisms of decoder-only language models:

- We show that language models can access their memory sequentially and can reproduce memorized content. In contrast, they encounter significant challenges in random memory access.

- We find solutions to mitigate the challenge of random access by permuting memorized content or explicitly reciting the memory before performing tasks.

- We demonstrate the effect of poor random memory access ability in open-domain question answering, showing that the challenge could have broader implications on the applications of language models.

2 Related Work

Memorization in Language Models. Large language models store a considerable amount of knowledge in their parameters (Petroni et al., 2019; Heinzerling and Inui, 2021). They memorize useful knowledge such as facts and commonsense (Zhao et al., 2023), but also sensitive personal information such as emails or phone numbers (Carlini et al., 2020; Huang et al., 2022). Existing approaches to understanding memorization include fine-grained analysis to locate the neuron that is associated with the knowledge (Meng et al., 2022; Liu et al., 2024) or macro analysis to understand the overall dynamics of memorization (Tirumala et al., 2022; Speicher et al., 2024). In this study, we do not aim to analyze the mechanisms of writing to language model’s memory. Instead, we consider the language model as a black-box memory store and focus mainly on how the model accesses its memory.

Knowledge Injection. Our investigation requires writing new content to the model’s parametric memory. There are mainly two ways to perform such knowledge injection without changing the model architecture (Ovadia et al., 2024; Balaguer et al., 2024): fine-tuning or retrieval augmentation. Retrieval augmentation (Lewis et al., 2020; Shi et al., 2023) retrieves relevant information and puts it into the model’s context while fine-tuning directly updates the model parameters. As the goal of our study is to investigate how the model accesses its parametric memory after writing to the memory, we choose finetuning as the method for introducing new knowledge to the model.

Knowledge Retrieval. Previous works have shown that using prompts can effectively retrieve knowledge stored in large language models (Bouraoui et al., 2019; Jiang et al., 2021; Wang et al., 2021). We follow earlier work to use prompts to query the model to access and regenerate memorized content. However, a notable difference is that prior work focuses on finding optimised methods to elicit the model’s knowledge obtained during pretraining (Youssef et al., 2023; Liu et al., 2023), while we directly use unique keys for memorizing and retrieving content.

Language Model as a Document Index. We consider the language model as a memory store for passages, which is related to the recent advances in adopting a language model as an index for document storage and retrieval (Metzler et al., 2021;
We abstract the language model as a memory bank (Tay et al., 2022; Wang et al., 2023; Zeng et al., 2023). In such indexes, each document is associated with a document identifier (ID), which could be keywords (Ren et al., 2023; Bevilacqua et al., 2022; Lee et al., 2023b,a) or numbers (Tay et al., 2022; Wang et al., 2023; Zhuang et al., 2022; Zhou et al., 2022). We also follow the practice and assign an ID to each document for storing and retrieving the documents. However, we do not ask the model to retrieve a relevant ID to a question. Instead, we provide the ID in the input, and investigate the possibility of sequentially or randomly accessing the corresponding document content.

3 Investigating Sequential and Random Memory Access

In this section, we investigate the ability of a language model to sequentially or randomly access its memory stored in the parameters. First, we provide formulations of language models serving as a memory bank of passages (§3.1). Within this framework, we define sequential memory access as the process of starting from the beginning of a memorized passage and progressively generating subsequent content. In contrast, we conceptualize random memory access as the model’s ability to initiate recall from any chosen location in a memorized passage and accurately regenerate the subsequent content. Based on these definitions, we first investigate the model’s sequential memory access ability by requiring it to recite full passages word by word (§3.2). Next, we test the random memory access ability of the model by asking it to recite selected sentences from memorized passages (§3.3). We further assess the model’s random access proficiency through a more challenging task involving question answering (§3.4).

3.1 Task Formulation

We abstract the language model as a memory bank and investigate its sequential or random access ability. We adopt a simple definition of a memory bank as a key–value store \( D = \{ k_i : p_i \} \), where \( k_i \) represents a unique identifier (ID) assigned to the content of the \( i \)-th passage.\(^1\)

There are two core functions that a memory bank needs to support: reading and writing.

Given that our memory bank is embodied as a language model, it is not straightforward to write

\[ \text{We test the sequential access ability of the language model by requiring it to recite full passages word by word (§3.2). Next, we test the random memory access ability of the model by asking it to recite selected sentences from memorized passages (§3.3). We further assess the model’s random access proficiency through a more challenging task involving question answering (§3.4).} \]

and read the model’s memory. Following previous work (Zhu and Li, 2023; Wang et al., 2021), for writing to the memory bank, we use fine-tuning to update the model’s parameters. For reading, we use prompting to elicit the model’s memory. Specifically, for each passage \( p_i \) with its corresponding identifier \( k_i \), we create two types of data instances: writing, \( S_{\text{write}}(k_i, p_i) \) and reading, \( S_{\text{read}}(k_i) \rightarrow p_i \), where \( S_{\text{write}} \) and \( S_{\text{read}} \) denote the prompts detailed in Appendix A.1.

As the primary goal of our study is to test whether the model can read (access) its stored content sequentially or randomly, we mainly vary the reading function across different experiments.

Given a corpus consisting of \( M \) passages, we split the corpus into two subsets: \( T \) training passages and \( V = M - T \) validation passages. We adopt a mixed training strategy as described by Zhu and Li (2023): During the training stage, we include \( S_{\text{write}} \) and \( S_{\text{read}} \) instances of \( T \) training passages, as well as \( S_{\text{write}} \) instances of \( V \) validation passages. Our objective is for the model to learn to associate each identifier with its passage content by training on the reading and writing instances of the training passages. During evaluation, we prompt the model with the \( S_{\text{read}} \) instances of the \( V \) validation passages to test the model’s memory access pattern.

3.2 Sequential Access: Full Recitation

We test the sequential access ability of the language model by asking it to reproduce the full passage content. Specifically, given an ID, the model is prompted to start from the beginning of the corresponding memorized passage and generate tokens consecutively. We evaluate the model’s performance to reproduce the content on the \( V \) validation passages, which requires the model to both memorize the passage content and sequentially access the memory with the provided key.

Setup. To investigate whether the model can handle identifiers and passage content of different types, we set \( T = 400 \) and \( V = 40 \) and consider the following variations. For the type of passage content \( p \), we examine two categories: (1) natural language (NL), comprising Wikipedia paragraphs from SQuAD (Rajpurkar et al., 2016), and (2) random strings (Rand), where each NL passage is substituted with a space-separated alphanumeric string maintaining the same number of tokens. Regarding the type of \( k \) (i.e., passage IDs), we explore

\[ \text{We use the term “document” and “passage” interchangeably in this paper, referring to a chunk of text.} \]
Table 1: BLEU/Exact Match scores of reading from memory with different types of IDs and passage content.

<table>
<thead>
<tr>
<th>Training Documents</th>
<th>BLEU</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
<tr>
<td>5k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
<tr>
<td>10k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
<tr>
<td>50k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
<tr>
<td>100k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
<tr>
<td>500k</td>
<td>96.7/95.0</td>
<td>73.4/72.5</td>
</tr>
</tbody>
</table>

Figure 2: EM and BLEU for reading validation passages, varying the number of training passages. We calculate EM using only the first 25 tokens, as the model often generates beyond the max passage length (25).

Table 3: Random Access: Selective Recitation

Selective recitation is a straightforward synthetic task: asking the language model to reproduce a specific sentence of a memorized passage. This task is designed for its simplicity, as it does not require the model’s understanding of passage content. The focus is solely on the model’s capacity to access segments in a memorized passage. Successful random access would be indicated by the model’s ability to reproduce any sentence from within memorized passages, regardless of position.

Setup. We follow Mallick et al. (2023) to place markers at the boundaries of each sentence, obtained by the NLTK sentence splitter:

```
[p]...[p]
```

where `p` is a passage identifier. For example, we may have `[0] sent0 [0] [1] sent1 [1]`, etc. In this case, the model only needs to learn to copy the content between these markers. Our selective recitation task requires the model to recite the `j`-th sentence of passage `p_i` based on the given passage ID `k_i`. The reading function is now

```
S_{read}(k_i,j) \rightarrow p_i[j],
```

such as “What is sentence [1] of Document #2033?”. For reference, we also test the model’s performance in a baseline where the passage content is provided in the context window.

As we are testing for exact memorization, we use BLEU and EM scores to evaluate the model. Similar to §3.2, we use `T = 400` training and `V = 40` validation passages, with 1994 sentences and 200 sentences respectively. We set the type of ID to be `Title` and only include passages with more than 3 sentences. All other hyperparameters stay the same.
Figure 3: A stacked bar plot showing the accuracy of ID-guided sentence recitation with different marker numbers. The performance decreases significantly as the sentence index grows.

the same as §3.2.

Discussion. We find that providing the passage ID does not enable the model to selectively recite the requested sentences. It scores poorly with a low EM of 34.5 and a 47.1 BLEU score, in contrast to the much higher 97.0 EM and 97.3 BLEU when the passage content is included in the context. A detailed analysis in Figure 3 reveals that the correct predictions are largely reciting the first sentence ($j = 0$). This verifies that the model can sequentially access the content to reproduce the first sentence. However, as the marker index increases, the model is required to skip preceding sentences and directly access a sentence in the middle of a passage. The model’s performance sharply declines, indicating its inability to randomly access middle or later sentences in memorized passages.

3.4 Random Access: Grounded Question Answering

Building on our earlier finding §3.2 that the model can memorize many passages each linked to a unique ID, we embark on a more pragmatic task: question answering grounded in a specific passage ID. This task aims to evaluate whether the model can provide answers to questions by extracting a span from its memory. For instance, a question might be framed as “According to document #3022, in what year did Chopin become a French citizen?” and the answer is “1835” in the passage with ID #3022. We hypothesize that if LMs are capable of random memory access, they should navigate to the corresponding passage using the provided ID and extract the relevant span to answer the questions.

Setup. We experiment with the well-known SQuAD-v1 (Rajpurkar et al., 2016) dataset because many of its questions are closely dependent on the passage, such as “How did the war start?”. This design compels the model to depend on the memorized IDs and passages rather than pre-existing knowledge. We explore the grounded QA task with variants of providing (1) the ID of the golden passage with the answer, (2) a random non-golden ID and (3) no ID. For comparison, we also consider the setups that do not involve writing passages to the model’s parametric memory. These include (1) closed-book QA, where the model is fine-tuned solely on QA pairs, serving as a baseline to assess the model’s reliance on prior knowledge for answering questions, and (2) open-book QA, where the golden passage content is concatenated with the question, setting the upper limit of extractive QA performance.

We experiment with different types of passage IDs. To ensure the uniqueness of using titles as passage IDs, we select $T = 442$ passages and $V = 48$ passages from the full SQuAD dataset, with over 2,000 and 300 questions respectively. The model is evaluated on F1 and EM following the original SQuAD evaluation script. The other hyperparameters are the same as mentioned in §3.2.

Discussion. The results are presented in Table 2 (the settings with “+Recitation” are discussed in later sections). As expected, the model performs the best in the open-book setting, as it only needs to locate the answer in the golden passage. In contrast, the closed-book QA setup yields the worst performance, as the model has no access to passages and relies solely on its parametric knowledge stored during pretraining.

Interestingly, the form of the provided passage ID has minimal impact on performance. We observe similar performance regardless of whether the golden ID is provided, except when the type of ID is Title. In this case, providing a random incorrect ID harms performance. We suspect that this is because the title is usually an entity related to the passage topic, therefore offering useful clues. In cases where the ID does not carry semantic meaning (i.e., Rare and Num), the correctness or presence of the ID does not significantly affect the performance, which remains substantially below the open-book setting, despite the model memorizing all passages. This further validates the model’s inability to effectively access random memory, as it struggles to extract the answer even when provided with a correct passage ID.
In summary, our findings validate the hypothesis that LMs can effectively function as a memory bank, enabling sequential access to its memory. However, there are significant limitations in the model’s ability to randomly access its memory. Across both the simple selective recitation and the complex grounded question-answering tasks, the model consistently fails to accomplish the tasks by leveraging its memory, despite being explicitly provided with the corresponding passage IDs.

4 Mitigating Random Access Challenge

The earlier experiments show that in general, language models perform well in sequentially accessing their parametric memory, but encounter challenges in random memory access. This naturally raises the question: How can we mitigate the shortcomings in random memory access?

4.1 Proposed Method

To address the challenge, we start from the two operations supported by LMs as a memory store: reading and writing. During the writing phase, we hypothesize that performing permutation on the passage content can naturally enhance the model’s random access ability: any part of the content can be the starting point of a memorized sequence. In this setup, we change the sequential order of passage content to achieve random access.

On the other hand, during the reading phase, leveraging the model’s context window presents a viable strategy. The attention mechanism (Vaswani et al., 2017) enables the model to access any token within the context window, thereby inherently supporting random access (Packer et al., 2023; Ge et al., 2023). For tasks with a given ID, we could ask the model to sequentially recite the passage first, place it within the context, and subsequently query the model to perform span extraction tasks utilizing this context, as illustrated in Figure 4. Our subsequent experiments are designed to evaluate the effectiveness of these two methods. Through empirical evaluation, we validate that content permutation during writing or recitation during reading can largely mitigate the challenge of random memory access and enhance performance.
Setup. We extend the earlier experiments by integrating recitation and permutation into the respective reading and writing stages.

First, we add a setup to the selective sentence recitation task: Based on the given ID, the model is tasked to first recite the entire content of the corresponding passage and then the specific sentence, altering the reading operation to \( S_{\text{read}}(k_i, j) \rightarrow (p_i, p_i[j]) \). Similarly, for the grounded QA task, we ask the model to recite the passage associated with the input passage ID before answering the question. In the setup without an ID, the model is still trained to recite the golden passage.

To explore the effect of permutation during the writing stage, we perform permutation among sentences in a passage to create diverse \( S_{\text{write}} \) instances. For a \( J \)-sentence passage we tested: (1) first, moving each sentence to the passage’s beginning to create \( J \) unique instances;

(2) random, randomly shuffling the sentences \( k \) times to create \( k \) instances, where \( k \) is set to 4 by default;

Discussion. Reciting the passage content effectively boosts the performance of selective recitation, as evidenced in Table 3. With recitation, the model first sequentially accesses the content from its memory using the provided passage ID and subsequently loads this passage in the context to allow for random access. Conditioned on the recited content in the context, the model can therefore easily identify the correct sentence.

Similarly, explicitly reciting the golden passages markedly enhances question-answering performance, as shown in Table 3. This observation is consistent across all three types of passage IDs. Conversely, intentionally prompting the model to recite a random passage leads to a decline in performance. This is likely because random passages introduce irrelevant information and confuse the model. Surprisingly, the recitation of relevant passages benefits performance even without an ID, although the improvement is smaller than with the golden ID. This verifies the effectiveness of recitation in more general settings of question answering.

Another way of enhancing random access is to perform permutation of sentences, as presented in Table 3. Simply bringing every sentence to the start of the passage once (first) or randomly permuting the sentences many times (random) helps to solve the challenge of accessing the middle content of a passage. We also find that permutation enhances ground performance (Appendix C) from 26.7 to 31.3 in terms of EM. However, it is noteworthy that permutation does not alter the inherent sequential access pattern of parametric memory. Rather, by permuting the sentences and disrupting their original order, we allow more sentences to be sequentially accessible via the ID.

5 Case Study: Open-Domain Question Answering

Our findings indicate that language models struggle with random memory access, unless the memory is explicitly recited and thus loaded into the context which can be accessed randomly. Building on this insight, we extend our study to the task of open-domain question answering, a challenging task that requires the model to first retrieve relevant memories and reason over them. This is different from previous experiments as the passage IDs are no longer provided as the input: The reading operation becomes \( S_{\text{read}}(q) \rightarrow \text{ans} \). The model therefore needs to find relevant passages to the query without the aid of passage IDs, which is a non-trivial task (Pradeep et al., 2023). As the goal of our study is not on retrieval performance and our earlier results (§3.3) show that the model has limited memorization capacity, we reduce the difficulty of retrieval by limiting the number of passages written to the model’s memory: we only include positive passages that contain answers to at least one question.

We aim to test the model’s ability to perform random access in real applications. Specifically, we investigate whether the model, having memorized many passages, can accurately extract answers from its memory. Similar to the previous experiments, we also aim to observe the difference in the model’s performance when it is trained to recite relevant passages and subsequently answer the question. We opt not to experiment with permutation due to the high training cost associated with sentence permutation across a large number of passages, and leave this avenue for future work.

5.1 Experimental Setup

We use Natural Questions (Kwiatkowski et al., 2019) processed by Karpuhin et al. (2020) for single-hop QA, selecting 6000 training and all of the 6489 validation questions, with a total of 10.9k passages. For multi-hop question answering, we use HotpotQA (Yang et al., 2018) where each question has two golden passages. We select 8k training
and all the 7405 validation questions in the distractor subset, with a total of 26.9k passages.

We start from a baseline setup where the training only involves QA pairs, i.e., closed-book QA. Next, we consider two types of training strategies to write the passages into the memory. In the mixed setting, the model is fine-tuned on a mixture of the $S_{write}$ instances of all passages and training QA pairs. In the continual setting, the model is fine-tuned on $S_{write}$ of all passages first, followed by fine-tuning on QA. To test the effectiveness of recitation, we also include settings where the model is trained to recite the golden passage(s) before answering. As the task requires the model to perform both passage retrieval and question answering, we expect that the model size should be sufficiently large. Therefore, we opt for GPT2-XL with 1.5B parameters. In the mixed setting, we train the model for 20 epochs with a learning rate of 3e-5. In the continual setting, we first train 20 epochs on the passages, followed by another 20 epochs on QA pairs. We report the best performance based on the EM score on validation questions.

### 5.2 Results and Discussion

The results presented in Table 4 demonstrate that writing golden passages into the model’s memory, with either mixed or continual training, leads to improved performance over the baseline closed-book setting. This aligns with our expectations, as we deliberately inject passages containing the answers to the questions into the memory, enriching the model’s knowledge.

Moreover, recitation significantly enhances the model’s ability to utilize and access memorized passages, leading to a noticeable improvement in performance. This is observed in both the mixed and continual training settings. The exact match score increases significantly by more than 3% in both single and multi-hop QA. When the model explicitly recites the passages and loads them into the context for random access, the original open-domain QA task is reduced to an easier task of extractive QA. However, the low recitation BLEU score suggests that the model does not always accurately recite the golden passage. We expect that the performance could be further enhanced if it can accurately retrieve relevant passages from memory.

The mixed training strategy outperforms the continual training setup. This is likely because the model’s memory of passage content is refreshed constantly in mixed training. In continual training, however, the second stage only involves QA pairs on training passages, potentially leading to fading memory of validation passages. Consequently, the recitation becomes less accurate, as shown by a decrease in the BLEU score.

Our results are consistent and complementary to the findings of Wei et al. (2023) and Sun et al. (2023): introducing intermediate steps or generating relevant passages helps to improve model performance on various tasks. We provide an alternative interpretation for this phenomenon: loading the parametric memory into the context window facilitates enhanced random access to memorized information, and the model benefits from such enhancements.

### 6 Conclusion

We empirically study how language models access their parametric memory. Our experiments on both synthetic and realistic data demonstrate that while language models can adequately reproduce memorized content in a sequential manner, they struggle with the random access of segments in the middle of memorized content. We identify two effective strategies of recitation and permutation to mitigate the limitation of random memory access. Furthermore, through a controlled case study on open-domain question answering, we illustrate that allowing the model to recite and ran-
domly access its memory significantly improves performance. Overall, our study not only provides a deeper understanding of memory access patterns in language models, but also highlights the implications of limited random memory access ability in practical application of language models.

**Limitation**

In this work, we mainly explore the memory access pattern of decoder-only language models of the GPT2 family. Future research is needed to understand whether our conclusions apply to other types of language models based on transformers such as encoder-only models and encoder-decoder models. Furthermore, we do not extend our study to larger models beyond 1.5B parameters due to computing resource constraints. It might be worthwhile to explore the scaling behavior of memory access patterns in larger language models. In addition, we mainly conduct controlled experiments on a text corpus of fixed size. Further investigation may be needed to explore how the findings can apply to large-scale pretraining corpus and their implications on pretrained language models.

**Ethical Considerations**

As the method suggests techniques to enhance access to the model’s memory, there could be malicious use of the recitation method to extract sensitive personal information from the model’s memory. We use open-source English datasets including questions and contexts from SQuAD-v1 (Rajpurkar et al., 2016), Natural Questions (Kwiatkowski et al., 2019), and Hotpot QA (Yang et al., 2018). We also use open-source English language models, GPT2, with different sizes (Radford et al., 2019). There might be potential biases in these datasets and models.

**Acknowledgements**

We appreciate the suggestions and comments by Do Xuan Long, Yanxia Qin, Yisong Miao, and other members in NUS WING. Tongyao Zhu is supported by the Industry PhD Program of Sea AI Lab.

**References**


Angels Balaguer, Vinamra Benara, Renato Luiz de Freitas Cunha, Roberto de M. Estevão Filho, Todd Hendry, Daniel Holstein, Jennifer Marsman, Nick Mecklenburg, Sara Malvar, Leonardo O. Nunes, Rafael Padilha, Morris Sharp, Bruno Silva, Swati Sharma, Vijay Aski, and Ranveer Chandra. 2024. RAG vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture.

Lucas Berglund, Meg Tong, Max Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2023. The reversal curse: LLMs trained on "A is B" fail to learn "B is A".


Yingqiang Ge, Yujie Ren, Wenye Hua, Shuyuan Xu, Juntao Tan, and Yongfeng Zhang. 2023. LLM as OS, agents as apps: Envisioning AIOS, agents and the AIOS-agent ecosystem. *ArXiv*, abs/2312.03815.


Oded Ovadia, Menachem Brief, Moshik Mishaei, and Oren Elisha. 2024. Fine-tuning or retrieval? comparing knowledge injection in LLMs.


Till Speicher, Afia Mohammad Khan, Qinyuan Wu, Vedant Nanda, Soumi Das, Bishwamitra Ghosh, Krishnna P. Gummadhi, and Evimaria Terzi. 2024. Understanding the mechanics and dynamics of memorisation in large language models: A case study with random strings.


A.1 Full Recitation

Given a key-value pair \( (k_i, p_i) \), the prompts are as follows:

\[
S_{\text{write}} = \text{"Article } k_i \text{, Content: } p_i \text{"}
\]

\[
S_{\text{read}}(k_i) \rightarrow p_i = \text{"Article } k_i : \text{ What is the content of this article?" } \rightarrow \{p_i\}
\]

A.2 Selective Recitation

In this experiment, we follow the same prompt of \( S_{\text{write}} \), as described in Appendix §A.1, and only change \( S_{\text{read}} \):

\[
S_{\text{write}} = \text{"Article } k_i \text{, Content: } p_i \text{"}
\]

\[
S_{\text{read}}(k_i) \rightarrow p_{i[j]} = \text{"Article } k_i : \text{ What is Sentence } \{j\} \text{ of this article?" } \rightarrow \{p_{i[j]}\}
\]

A.3 Grounded Question Answering

In this experiment, we follow the same prompt of \( S_{\text{write}} \), as described in Appendix §A.1, and only change \( S_{\text{read}} \) to questions related to \( p_i \). \( S_{\text{read}}(k_i, q) \) represents the instances where the recitation of the passage content is prepended before the answer.

\[
S_{\text{write}} = \text{"Article } k_i \text{, Content: } p_i \text{"}
\]

\[
S_{\text{read}}(k_i, q) \rightarrow \text{ans} = \text{"Article } k_i \text{ in Question: } q \text{ in Answer: } \rightarrow \{\text{ans}\}
\]

\[
S_{\text{read}}(k_i, q) \rightarrow (p_{i, ans}) = \text{"Article } k_i \text{ in Question: } q \text{ in Answer: } \rightarrow \{p_{i}\} || \text{Answer: } \{\text{ans}\}
\]

A.4 Open-Domain Question Answering

In the setup of open-domain question-answering experiments, we no longer have a pre-assigned ID for each document. Our \( S_{\text{write}} \) becomes:

\[
S_{\text{write}}(p_i) = \text{"Document: } p_i \text{"}
\]

Similarly, the reading operation now does not have any ID associated with it, but only a question. It becomes:

\[
S_{\text{read}}(q) \rightarrow \text{ans} = \text{"Question: } q \text{ in Answer: } \rightarrow \{\text{ans}\}
\]
In the case of recitation, our prompts for training the model include the passage containing the answer.
\[ S_{\text{rec}}(q) \rightarrow (p_{\text{golden}}, \text{ans}) = \text{“Question: \{q\}”} \rightarrow \text{“Related documents: \{p_{\text{golden}}\} \& Answer: \{\text{ans}\}”} \]

### B Additional Selective Recitation Experiments

We provide additional experimental results for our selective recitation task of reciting sentences. All of the experiments lead to a consistent conclusion that the model is unable to randomly extract a sentence from a memorized passage.

In both of the experiments below, we include setups of (1) in-context: the passage is included in the context window. (2) ID-guided: the basic version of the selective recitation task where a passage ID is provided. and (3) with passage recitation: the passage is recited first before sentence recitation.

#### B.1 Reciting the first/second/last sentence

As a basic setting of the selective sentence recitation task, we ask the model questions like “What is the [first/second/last] sentence of Article #123?”.

The results are shown in Table 5. The model almost always recites the first correctly, while recitation performance drops significantly for the second or last sentence. This shows that the model is performing sequential access: following the article ID, the model can only access content immediately after the ID – the first sentence. It is unable to directly access the second or last sentence.

We observe that even for the in-context setting where the passage is in the context window, the model does not perform perfectly, especially for extracting the last sentence. This is because the model also needs to learn what first, second or last means, which involves numerical reasoning ability to count the index. Therefore, in the main experiments, we put markers on both ends of a sentence to reduce the task difficulty.

#### B.2 Reciting the next/previous sentence

We perform experiments to find the sentence before and after an input sentence in a given passage. In other words, our \( S_{\text{read}} \) operation becomes \( S_{\text{read}}(k_i, s_j) \rightarrow s_{j+1}/j-1 \), where \( s_j \) is the input sentence. The results are shown in Table 6.

We observe that finding the sentence after the input sentence is always easy, while the reverse task is much more difficult. This also reveals that the model reads its memory sequentially. It is unable to randomly access the sentence before the input \( s_j \), even if the target sentence is adjacent to the input sentence.

### C Additional Grounded Question Answering Experiments with Permutation

We conduct additional experiments showing the effect of performing permutations of sentences of memorized passages in the grounded question-answering task.

In Table 7, we show the effect of augmenting the \( S_{\text{write}} \) instances to include sentence permutations of the original passage. We observe that the model’s performance generally improves as we perform permutation. This validates our findings that performing permutation enhances random access to the passage content.

### D Additional Open-Domain Question Answering Experiments

To ensure that our conclusion is consistent with different dataset sizes, we vary the number of training and validation documents and questions to observe the performance difference. For NQ, we select 5k training and 5k validation QA pairs, forming a corpus containing around 9k passages. For Hotpot QA, we select 5k training and 5k validation questions in the distractor subset, with a total of 18.2k passages.

In Table 8, we obtain similar conclusions that recitation greatly enhances question-answering performance, and using a mixed training strategy is better than continual training because of the increase in recitation score.

### E Additional Training Details

We conduct all experiments in a cluster with NVIDIA Tesla A100 GPUs (with 40G or 80G memory). Experiments in §3.2 take a total of 48 hours on 4 GPUs. Selective sentence recitation experiments in §3.3 and §4 take a total of 41 hours on 4 GPUs. Grounded QA experiments take a total of 132 hours on 4 GPUs. The open-domain QA experiments need 3 days to complete with 32 GPUs.

We use the Huggingface transformers library for all experiments. We use a learning rate of 3e-5. We set a constant learning rate schedule for the open-domain QA experiments. For all other experiments,
we use a warmup ratio of 0.05 and a linear decay learning rate. We evaluate the model’s performance on the validation set at the end of each epoch.