Peripheral Memory for LLMs: Integration of Sequential Memory Banks with Adaptive Querying

Songlin Zhai [1] **Yuan Meng** [1] **Yongrui Chen** [1] **Yiwei Wang** [2] **Guilin Qi** [1]

Abstract

Large Language Models (LLMs) have revolutionized various natural language processing tasks with their remarkable capabilities. However, a challenge persists in effectively processing new information, particularly in the area of long-term knowledge updates without compromising model performance. To address this challenge, this paper introduces a novel memory augmentation framework that conceptualizes memory as a peripheral component (akin to physical RAM), with the LLM serving as the information processor (analogous to a CPU). Drawing inspiration from RAM architecture, we design memory as a sequence of memory banks, each modeled using Kolmogorov-Arnold Network (KAN) to ensure smooth state transitions. Memory read and write operations are dynamically controlled by query signals derived from the LLMs' internal states, closely mimicking the interaction between a CPU and RAM. Furthermore, a dedicated memory bank is used to generate a mask value that indicates the relevance of the retrieved data, inspired by the sign bit in binary coding schemes. The retrieved memory feature is then integrated as a prefix to enhance the model prediction. Extensive experiments on knowledge-based model editing validate the effectiveness and efficiency of our peripheral memory.

1. Introduction

Large Language Models (LLMs) have transformed the field of machine learning, achieving state-of-the-art performance across a wide range of tasks (Singh, 2023; Naveed et al., 2024; Chen, 2024; Luo et al., 2024; Azaria et al., 2024). Despite these remarkable advancements, LLMs still face challenges in integrating *new information* (Wang et al., 2024b; Modarressi et al., 2025), particular in incorporating *knowledge updates* without harming their original performance (Zhang et al., 2024; Fang et al., 2025). These challenges are further exacerbated in dynamic real-world applications (Kaddour et al., 2023; Wang et al., 2024b). Memory augmentation has emerged as a promising solution to address these challenges (Zheng et al., 2023; Hartvigsen et al., 2023; Wang et al., 2024a;b; Modarressi et al., 2025), with existing efforts focusing on *working memory* (contextual key-values pairs from specific layers), *implicit memory* (extra model parameters), and *explicit memory* (sparsely activated knowledge circuits) (Yang et al., 2024). However, as summarized in Table 1, these methods are hindered by limitations in **scalability** (*i.e.,* maintaining query effectiveness and efficiency as memory size increases), **reusability** (*i.e.,* sharing memory across LLMs with varying architectures to avoid redundant storage of identical knowledge), and **configurability** (*i.e.,* dynamically adjusting memory independently of the LLM, such as memory size and structure). These issues primarily stem from the tight coupling between memory and LLM architectures, which restricts dynamic memory allocation and cross-model adaptability.

Table 1. Comparison between different types of memory. *W.M.*: "Working Memory", *I.M.*: "Implicit Memory", *E.M.*: "Explicit Memory", ***P.M.***: "Peripheral Memory (Ours)". 🔋: "Well Supported", 🪫: "Partially Supported", ▭: "Poorly Supported".

Memory Types	*W.M.*	*I.M.*	*E.M.*	***P.M.***
Scalability	🪫	🪫	🪫	🔋
Reusability	▭	▭	▭	🔋
Configurability	▭	▭	▭	🔋

Inspired by modern computer architectures that separate computation and storage (Blum, 1986; Rojas & Hashagen, 2000), we propose a novel memory framework called **peripheral memory**. Unlike traditional memory systems, we decouple memory from LLMs, treating it as an independent module-hence the term "*peripheral*". In this framework, the LLM functions as the information processor, similar to a CPU, while the peripheral memory operates like RAM, handling requests from LLMs to retrieve or store information

[1]School of Computer Science and Engineering, Southeast University, Nanjing, China [2]University of California, Merced, USA. Correspondence to: Guilin Qi <gqi@seu.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

as needed. This decoupling offers two key advantages: the flexibility to design memory independently of the LLM's architecture, and the ability to treat memory contents as dynamic variables, free from constraints imposed by the LLM (*e.g.,* hidden feature dimension, depth of the model).

For the memory architecture, we draw inspiration from physical RAM (Random Access Memory) (Jacob et al., 2007), structuring the peripheral memory as a set of memory banks, each functioning as a one-bit data unit. The number of memory banks thus determines the overall memory bandwidth, enabling flexible scaling of memory capacity. For each memory bank, we conceptualize its query process as a mapping from query signals to memory data, and employ a Kolmogorov-Arnold Network (KAN) (Liu et al., 2024) to model this process, due to recognizing KAN's ability to fit nonlinear interactions with fewer parameters. This could ensure efficient and precise memory retrieval while minimizing computational overhead. Moreover, instead of working independently, each memory bank refines its query iteratively, passing its output to the next one. This design, termed *sequential*, allows memory banks to process information sequentially, promoting smoother integration of information and enhancing the handling of interdependent queries.

For memory querying, if the required data is not stored in memory, the query results are likely to be useless or even detrimental to the LLM. In this case, directly integrating the query results into the LLM could disrupt its forward accumulation of information. To ensure query reliability, we introduce a confidence memory bank that assesses the relevance of retrieved data by generating a trust value derived from the sequential outputs of previous banks. This mechanism filters out irrelevant or noisy information, ensuring that only relevant memory content is integrated into the LLM, thereby preserving its original knowledge and preventing interference. Memory writing is implemented through a fine-tuning process, eliminating the need for complex manual operations and improving usability. Furthermore, when memory reaches capacity, older data is archived, and new data is stored in fresh memory, enabling efficient long-term memory management. Extensive experiments on knowledge editing validate the effectiveness of the proposed peripheral memory. In summary, the contributions of this paper are:

- We propose a novel peripheral memory for the LLM that decouples the memory module from the model itself. This design alleviates the limitations of existing memory paradigms, enhancing the scalability, reusability, and configurability of the memory system.
- We introduce a sequential querying mechanism, where the output of each memory bank serves as the input to the next. Additionally, a confidence memory bank is allocated to generate trust values, minimizing interference with the LLMs' original knowledge.
- We conduct extensive experiments on Knowledge-based Model Editing, demonstrating the effectiveness of the proposed memory in improving task performance and efficiency, along with enhanced knowledge storage management.

2. Preliminaries

2.1. Notations and Task Definition

A large language model requires a textual sentence s as input, which is tokenized into a sequence of tokens $s \rightarrow \{t_0, t_1, ..., t_n, ..., t_N\}$. Each token is converted into a corresponding embedding vector which is processed by each hidden layer of the model. We use the bold notation $\boldsymbol{t}_n^l$ to represent the hidden representation of t_n at layer l, where $1 \leq l \leq L$ indexes the layers of the LLM. Additionally, we denote the memory as $\mathcal{M} = \{M_1, M_2, ..., M_k, ..., M_K\}$, with M_k being an individual memory bank (a specific storage unit). To query the memory, a query signal $\boldsymbol{q}$ is used. After querying, the resulting memory data is denoted as $\boldsymbol{m} = (m_1, m_2, ..., m_k, ..., m_K)$, where m_k is the query result from k-th memory bank (*i.e.,* M_k). For augmenting LLM with memory, the task can be formulated as:

$$f_{\boldsymbol{\Theta}}(x|\mathcal{M}) = \begin{cases} y^*, & \mathcal{K}(x) \in \mathcal{M} \\ y, & \mathcal{K}(x) \notin \mathcal{M} \end{cases} \tag{1}$$

where $\boldsymbol{\Theta}$ denotes the LLM parameters, and f is a complex function learned by the LLM. $\mathcal{K}(x)$ represents the relevant knowledge related to input x. If the relevant information is stored in the memory, *i.e.,* $\mathcal{K}(x) \in \mathcal{M}$, the LLM is expected to output the desired prediction y^*. Otherwise, the model should maintain its original prediction y.

2.2. Architecture of Physical RAM

Physical RAM is a critical component in modern computer systems, providing high-speed temporary storage for data. The architecture of RAM typically consists of one or two *memory ranks*. Each *memory rank* is composed of several *memory chips*, and each chip contains multiple *memory banks*. Specifically, a *memory bank* refers to a memory array for storing data, and is indexed by specific row and column identifiers. Each *memory cell* within a bank, identified by a unique (*row_id, column_id*), stores specific bits of data and is predefined by the manufacturer.

For the memory query process, we can conceptualize it as a mapping: $X \mapsto \mathcal{Y}$, where X represents the query signal from the CPU, and $\mathcal{Y}$ is the corresponding sequence of 0 and 1 returned from memory. Specifically, each bit of the query can thus be modeled as:

$$\phi_k(X|\mathbb{M}) = \mathbb{I}(X|\mathbb{M}) \tag{2}$$

where ϕ_k indicates the query function of k-th bit in RAM

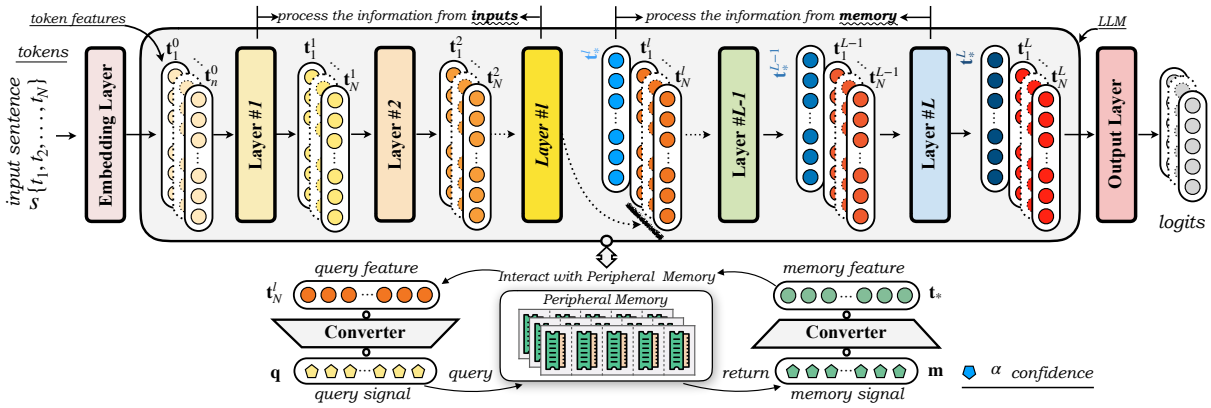

Figure 1. Framework of peripheral memory. Reading&Writing operations are achieved by the query signal from LLM hidden states.

$\mathbb{M}$. $\mathbb{I}$ is an indicator function that outputs either 0 or 1. In this context, $\mathbb{I}(X|\mathbb{M})$ represents the presence or absence of the desired bit of information from the physical RAM $\mathbb{M}$ based on the query signal X.

3. Methodology

Different from previous memory architectures that directly store information within the LLM, our peripheral memory decouples the memory module from the LLM, treating it as an independent storage component. In this design, the content stored in memory is treated as a dynamic variable, allowing the LLM to dynamically retrieve or write the necessary information as needed. Figure 1 illustrates the overall framework. The memory is connected to the LLM through a specialized converter, which acts like signal cables to facilitate feature conversion between the two components. Memory operations are driven by the hidden state of the last token from the LLM (serves as the query signal), and the query result is then integrated into the LLM as a prefix, enhancing the generation process.

3.1. Element-Wise Peripheral Memory

3.1.1. Structure of Memory

Similar to physical RAM, our peripheral memory is composed of a set of distinct memory chips. However, unlike physical RAM, we simplify the memory architecture by assigning each memory chip to contain a single memory bank, with each bank representing a one-bit data unit[1]. Figure 2 depicts the detailed architecture of the proposed peripheral memory. In line with Eq. 2, the query of each memory bank (*e.g.,* M_k) is conceptualized as a mapping from the query signal to the query result, formulated as:

$$m_k = M_k(q_k) \tag{3}$$

[1]The terms *memory chip* and *memory bank* are equivalent in peripheral memory, as each chip contains only one memory bank.

where m_k is the memory data at k-th element in the memory feature vector $\boldsymbol{m}$, retrieved by q_k. Specifically, q_k is also the k-th element in the query signal $\boldsymbol{q}$.

Unlike the discrete bits in RAM, which can be understood as indicator functions, the query function in peripheral memory is designed to be continuous and smooth to avoid information loss. To model this smooth function for each memory bank (also denoted as M_k for simplicity), we employ Kolmogorov-Arnold Networks, which are capable of modeling complex, non-linear relationships in continuous spaces with fewer parameters (Kolmogorov, 1957; Liu et al., 2024). Formally, the memory bank M_k is defined as:

$$M_k(q_k) = \sum\nolimits_{i=1}^{D_k} g_i^k(h_{i1}^k(q_k)) \tag{4}$$

where D_k is the memory depth of M_k, which can be configured freely. g_i^k and h_{i1}^k are simple one-variable continuous functions parameterized by B-spline function (Aziznejad & Unser, 2019; Bohra et al., 2020; Liu et al., 2024).

3.1.2. Sequential Querying Mechanism

As shown in Eq. 3, each memory is independently retrieved by its corresponding query signal q_k. To enable richer interactions between memory banks and allow the memory system to capture complex patterns in query data, we introduce a sequential querying mechanism. That is, the output of one memory bank serves as the input to the next. As such, the querying process in Eq. 3 can be extended as:

$$m_k = M_k(m_{k-1}, q_k), \quad 2 \le k \le K \tag{5}$$

where m_{k-1} denotes the query result of the last memory bank M_{k-1}. From Eq. 5, we can draw that the peripheral memory contains a total of K memory banks to store data, with the first memory bank accepting only the query signal (formulated by Eq. 3), and subsequent memory banks accepting both the query signal and the query result from the

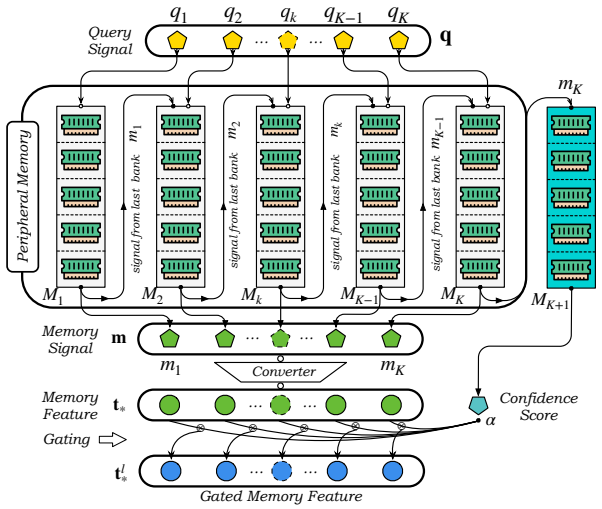

Figure 2. Architecture of the peripheral memory.

previous bank. For the memory bank where $2 \leq k \leq K$, Eq. 4 can be redefined as:

$$M_k(m_{k-1}, q_k) = \sum_{i=1}^{D_k} g_i^k(h_{i1}^k(q_k) + h_{i2}^k(m_{k-1})) \quad (6)$$

where $h_{i2}(m_{k-1})$ represents the processing of information from the previous memory bank (M_{k-1}), also parameterized by the B-spline function.

3.1.3. Confidence Memory Bank

To ensure the quality of the retrieved information and prevent disturbances to the LLM's existing knowledge, we introduce a confidence memory bank, represented as M_{K+1} in Figure 2. This confidence memory bank receives the memory states output by the previous memory bank (M_K) and calculates a trust value to assess the relevance of the data retrieved from memory. This value reflects the confidence level in the retrieved memory and acts as a mask to filter and refine the retrieved information before it influences the LLM's processing pipeline. The confidence value α predicted by the M_{K+1} memory bank is formulated as:

$$\alpha = M_{K+1}(m_K) \quad (7)$$

Analogously, the M_{K+1} memory bank can be modeled as $\sum_{i=1}^{D_{K+1}} g_i^{K+1}(h_{i1}^{K+1}(m_K))$. The masking operation is particularly useful in **Knowledge Updates**: where new, relevant knowledge should be prioritized over outdated data.

3.2. Memory Reading & Writing

Before reading from a peripheral memory, we first *plug it into* an LLM. Then, a query feature from the LLM is used to retrieve the memory, defined as:

$$\boldsymbol{t}_*^l = \sigma(\alpha) \cdot \boldsymbol{t}_* = \sigma(\alpha) \cdot \{\mathcal{M}((\boldsymbol{t}_N^l)^\top \boldsymbol{W}_0)^\top \boldsymbol{W}_1\} \quad (8)$$

where $\boldsymbol{t}_N^l$ is the hidden state of the last token at layer l. $\boldsymbol{W}_0$ converts the token feature into the query signal, acting as the *outlet cable*. Similarly, $\boldsymbol{W}_1$ maps the memory data $\boldsymbol{m}$ into the memory feature adapted to LLM hidden feature space, which can be understood as the *leading in cable*. The confidence value α is normalized by the sigmoid function σ, and then is used to mask the memory feature element by element. $\boldsymbol{t}_*^l$ is the masked memory feature, which is merged as a prefix vector into the LLM. Eq. 8 allows the memory and the LLM to be used effectively as a unified system. As such, the memory writing could be simply achieved by performing a finetuning process with setting requiring gradients of the memory. After the peripheral memory is "*full*", we can unplug and archive it and store data in a fresh one[2]. The memory bandwidth K plays a critical role in determining the read/write performance of the peripheral memory. To ensure K operates within an optimal range, we leverage the *Information Abundance* in Guo et al. (2024) to determine that its lower bound is approximately 64.

4. Experiments

In this section, we evaluate the proposed peripheral memory within a representative scenario involving knowledge updates, specifically Knowledge-based Model Editing. We primarily focus on analyzing its strengths in terms of scalability, reusability, and configurability, highlighting its advantages compared to existing memory architectures.

4.1. Knowledge-based Model Editing (KME)

4.1.1. Experimental Setup

The KME task evaluates the ability of the proposed peripheral memory to integrate specific knowledge updates into an LLM while ensuring that its original information remains unaffected. Following prior studies (Mitchell et al., 2022a; Meng et al., 2022; 2023; Li et al., 2024), we also utilize the EDIT sets of **ZsRE** (Levy et al., 2017) and **CounterFact** (Meng et al., 2022) to comprehensively evaluate all competing methods. To assess editing performance, we adopt three fundamental metrics, *i.e.,* **Efficacy**, **Generality**, and **Locality**. Notably, we consider the *consecutive editing*, where a series of sequential updates is performed without parameter roll-back. This setting effectively tests memory's capability to manage long-term knowledge updates.

All experiments are conducted on an *NVIDIA A100-SXM4-40GB* machine. The baseline methods are implemented using the widely adopted EasyEdit toolkit[3], ensuring reproducibility, reliability, and adherence to established research practices. Hyperparameter settings for all baseline models are based on recommendations provided by EasyEdit.

[2] Section 4.4 discusses memory storage capacity.

[3] https://github.com/zjunlp/EasyEdit

Table 2. Editing performance of all compared methods under $3K$ ***consecutive editing***. For our method, query features are derived from the hidden states (the last token representation) of the model's 24-th layer.

Editor	ZsRE				CounterFact			
	Efficacy	*Generality*	*Locality*	*Score*	*Efficacy*	*Generality*	*Locality*	*Score*
GPT-J (6B) (Wang & Komatsuzaki, 2021)	0.2165	0.2110	/	0.2138	0.0030	0.0023	/	0.0027
FT-L (Zhu et al., 2020)	0.1104	0.0841	0.0159	0.0701	0.2133	0.0797	0.0127	0.1019
LoRA (Hu et al., 2022)	0.0111	0.0115	0.0005	0.0077	0.0097	0.0067	0.0013	0.0059
ROME (Meng et al., 2022)	0.3187	0.2810	0.1829	0.2609	0.0013	0.0020	0.0003	0.0012
R-ROME (Gupta et al., 2024a)	0.5478	0.5176	0.1333	0.3996	0.6927	0.3740	0.4187	0.4951
MEMIT (Meng et al., 2023)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
AlphaEdit (Fang et al., 2025)	0.0000	0.0000	0.0005	0.0002	0.0000	0.0000	0.0000	0.0000
PMET (Li et al., 2024)	0.0002	0.0002	0.0003	0.0002	0.0000	0.0000	0.0000	0.0000
KN (Dai et al., 2022)	0.0000	0.0000	0.0002	0.0001	0.0001	0.0000	0.0000	0.0000
EMMET (Gupta et al., 2024b)	0.5521	0.5167	0.3747	0.4812	0.7020	0.4117	0.3303	0.4813
GRACE (Hartvigsen et al., 2023)	0.3368	0.0126	1.0000	0.4498	0.0002	0.0000	0.9939	0.3314
IKE (Zheng et al., 2023)	0.9973	**0.9898**	0.4764	0.8212	0.9900	**0.4223**	0.6393	0.6839
WISE (Wang et al., 2024a)	0.4598	0.4185	0.9895	0.6226	0.3420	0.0907	0.0790	0.1706
Ours (without $1K$ archive)	0.9934	0.6619	**1.0000**	0.8851	0.9240	0.1647	**1.0000**	0.6962
Ours (with $1K$ archive)	**0.9985**	0.6941	**1.0000**	**0.8975**	**0.9907**	0.2340	**1.0000**	**0.7416**
Llama 3 (8B) (Llama Team, 2024)	0.2627	0.2598	/	0.2613	0.0087	0.0075	/	0.0081
FT-L (Zhu et al., 2020)	0.0769	0.0666	0.0069	0.0501	0.0575	0.0047	0.0013	0.0212
LoRA (Hu et al., 2022)	0.1145	0.1116	0.0535	0.0932	0.0077	0.0117	0.0017	0.0070
ROME (Meng et al., 2022)	0.0339	0.0280	0.0015	0.0211	0.2507	0.1323	0.0097	0.1309
R-ROME (Gupta et al., 2024a)	0.0271	0.0243	0.0035	0.0183	0.4892	0.3662	0.0147	0.2900
MEMIT (Meng et al., 2023)	0.0000	0.0000	0.0396	0.0132	0.0000	0.0000	0.0722	0.0241
AlphaEdit (Fang et al., 2025)	0.0001	0.0000	0.0003	0.0001	0.0033	0.0017	0.0007	0.0019
PMET (Li et al., 2024)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
EMMET (Gupta et al., 2024b)	0.0517	0.0486	0.0043	0.0349	0.5450	**0.3882**	0.0128	0.3153
GRACE (Hartvigsen et al., 2023)	0.0624	0.0095	1.0000	0.3573	0.0003	0.0000	0.9938	0.3314
IKE (Zheng et al., 2023)	0.5233	0.5231	0.5289	0.5251	0.0055	0.0043	0.6509	0.2202
WISE (Wang et al., 2024a)	0.3348	0.3283	0.9997	0.5543	0.1473	0.0763	0.9907	0.4048
Ours (without $1K$ archive)	0.9597	0.5619	**1.0000**	0.8405	0.9038	0.2168	**1.0000**	0.7069
Ours (with $1K$ archive)	**0.9805**	**0.6123**	**1.0000**	**0.8643**	**0.9915**	0.3108	**1.0000**	**0.7674**

4.1.2. Overall Comparison

Table 2 summarizes the performance of all compared methods under $3K$ consecutive updates. The results highlight several key observations **(1)** Our method significantly outperforms all baselines across most metrics, achieving state-of-the-art results. Notably, even when compared to the latest memory-based KME methods, such as GRACE, IKE and WISE, our method demonstrates a significant performance advantage. This underscores the scalability and robustness of our framework, particularly in handling large-scale memory updates. Additionally, our method consistently achieves perfect performance on *Locality*, benefiting directly from our novel memory architecture, the "*peripheral*" memory. Specifically, when original predictions from the model are desired, the peripheral memory can simply be "*unplugged*" without causing any modifications to the original model. **(2)** While EMMET and IKE achieves slightly better performance in the *Generality* on CounterFact, its *Locality* score is considerably lower than that of ours. This suggests that EMMET introduces substantial disruptions to the original model. **(3)** Localization-based methods, such as ROME, AlphaEdit and PMET, suffer severe performance degradation after consecutive updates, with their scores eventually dropping to zero. This indicates that these methods not only fail to retain newly edited knowledge but also introduce significant interference, severely compromising the original model's integrity. As a result, they are unsuitable for large-scale or long-term knowledge editing.

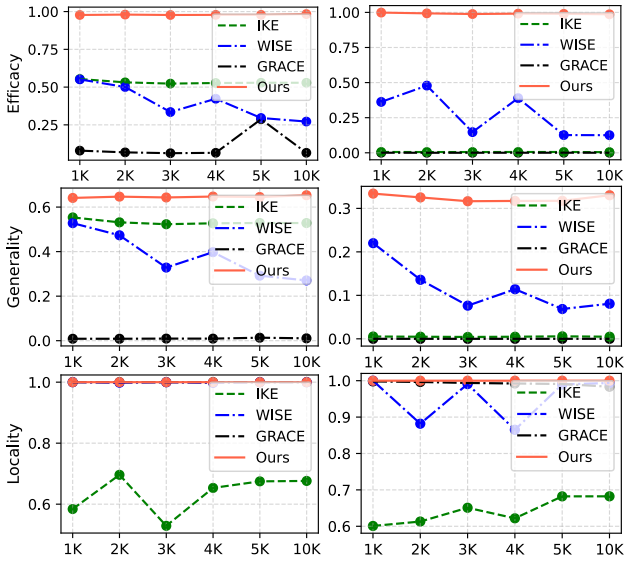

Figure 3. Performance of long-term knowledge updates on ZsRE (left) and CounterFact (right) based on Llama3 (8B). Our method is evaluated under $1K$ archiving.

Table 3. Reusability of peripheral memory. Memory writing is performed on Llama3 (8B), after which it is shared with other LLMs for reading only. The values labeled with † represent performance with shared memory, while those without † indicate performance with fresh memory written directly on the LLM itself.

Models	*Efficacy*	*Generality*	*Locality*	*Score*
		ZsRE		
Llama3 (8B)	0.9907	0.6090	1.0000	0.8666
Gemma2-it (2B)	0.9918	0.6989	1.0000	0.8969
	0.9713†	0.6635†	1.0000†	0.8783
Phi3 (3.8B)	1.0000	0.6547	1.0000	0.8849
	0.0000†	0.0000†	1.0000†	0.3333
		CounterFact		
Llama3 (8B)	0.9990	0.3150	1.0000	0.7713
Gemma2-it (2B)	0.9890	0.1948	1.0000	0.7279
	0.9790†	0.2045†	1.0000†	0.7278
Phi3 (3.8B)	0.9997	0.2875	1.0000	0.7624
	0.9997†	0.3075†	1.0000†	0.7691

4.1.3. Scalability: Long-Term Retention

To further investigate the scalability of our method, we conduct experiments by increasing the number of reads and writes in memory from $1K$ to $10K$, and analyzing the effects of long-term editing. Figure 3 presents the results of our methods compared to several strong memory-based methods across varying numbers of edits. The results demonstrate that our peripheral memory effectively adapts to a larger number of memory updates without experiencing performance degradation. In contrast, WISE (implicit memory) exhibits a sharp decline in efficacy and generality as the number of edits increases. Meanwhile, GRACE and IKE (working memory) exhibit a more stable trend but consistently underperform compared to our method. This lower performance is likely due to noise accumulation in working memory, which affects retrieval accuracy over time.

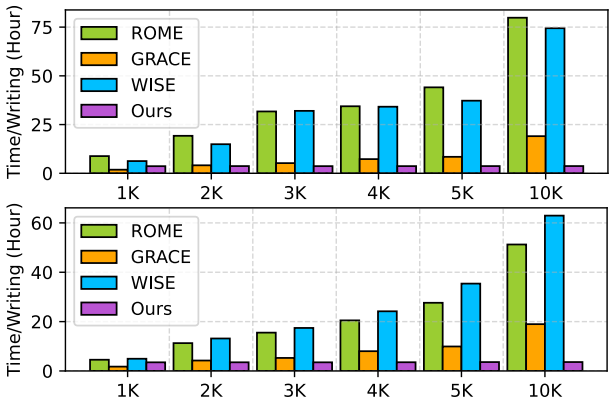

Figure 4. Writing efficiency of extensive knowledge updates on ZsRE (top) and CounterFact (bottom) based on Llama3 (8B). Our method is evaluated using parallel storage of a $1K$ archiving.

4.1.4. Scalability: Storage Efficiency

In scenarios involving a large number of edits, both performance and storage efficiency are critical considerations. Figure 4 illustrates the storage efficiency of our method compared to others under extensive editing conditions. From the $1K$ update results, we observe that while our model's write latency falls between GRACE (working memory) and WISE (implicit memory), it benefits from high configurability (see Section 4.4), enabling efficient parallel writes. The results ranging from $1K$ to $10K$ further illustrate that, owing to its parallel storage capability, our method maintains consistently stable and low efficiency values across scales. Among the baselines, WISE exhibits the longest write time since it internalizes knowledge directly into model parameters, whereas GRACE achieves the shortest latency due to its lightweight working memory mechanism. Additionally, ROME, despite being an efficient localization-based approach, takes longer than memory-based methods.

4.1.5. Reusability: Memory Sharing

As discussed in Section 3.1, the proposed peripheral memory is entirely independent of the underlying LLM and operates with a high degree of autonomy. This independence endows our memory with a significant advantage, *i.e.,* reusability. Specifically, knowledge stored in one LLM's peripheral memory can be seamlessly shared with other LLMs, elim-

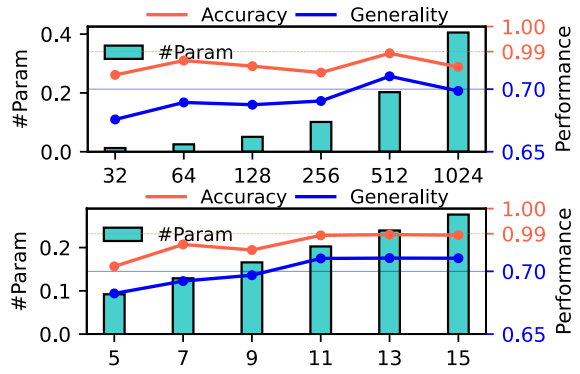

Figure 5. Effects of memory bandwidth (top) and depth (bottom) based on Gemma2-it (2B) and ZsRE with $1K$ consecutive editing.

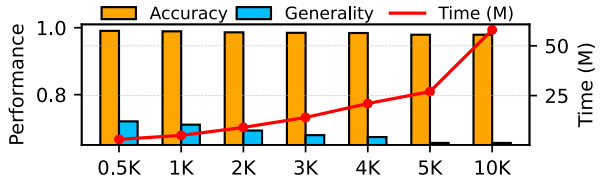

Figure 6. Storage capacity with bandwidth 512 and depth 11 based on Gemma2-it (2B) and ZsRE under consecutive editing.

inating the need for redundant knowledge storage. This follows the principle: *store once, use everywhere*. Table 3 presents the results of $1K$ memory sharing, where knowledge is first stored using Llama3 (8B) and then transferred to two different LLMs[4]: Gemma2-it (2B) and Phi3 (3.8B). The results demonstrate that even when memory is shared across LLMs with different architectures, knowledge retention remains consistently high, achieving nearly identical effectiveness as knowledge stored from scratch. In contrast, existing memory architectures do not support cross-model reuse due to constraints such as incompatible feature dimensions, parameter spaces, and internal representations. Notably, the performance of Phi3 under shared memory of ZsRE is 0. The possible reason is that ZsRE, being a zero-shot relation extraction dataset, relies more on the LLM's internal knowledge rather than externally stored memory.

4.2. Query Features from Different Hidden Layers

In this paper, we directly utilize hidden-layer representations as query features. However, selecting features from different layers can significantly impact memory performance. Figure 7 illustrates memory performance of *Accuracy* and *Generality*, under $3K$ consecutive edits (with a $1K$ archive). As depicted, employing features from earlier layers (*i.e.,* layers 1–7) as queries results in poor accuracy and generality, mainly because these layers predominantly handle low-level

[4]We utilize the hidden states from the 24-th layer of Gemma2 and the 28-th layer of Phi3 as the query features.

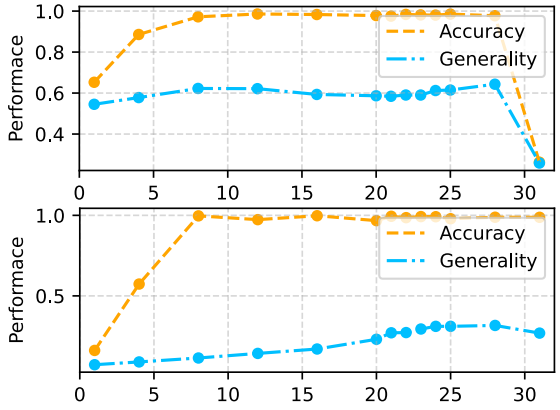

Figure 7. Performance of queries from different layers within Llama3 (8B) on ZsRE (top) and CounterFact (bottom).

linguistic information, thereby lacking sufficient semantic integration in the final token representations. As we move the query to higher layers, performance on both metrics gradually improves and ultimately stabilizes. However, for very late layers (*i.e.,* layers 30–32), the model's performance on ZsRE experiences a marked decline. Therefore, to maintain consistently stable and high performance, we select the 24-th layer as our query layer throughout this paper.

4.3. Memory Configurability

4.3.1. Memory Bandwidth

To evaluate the effects of memory bandwidth, we vary the number of memory banks from 32 to 1024 (*i.e.,* the number of parallel storage units) while keeping the memory depth fixed at 11. As shown in the top section of Figure 5, increasing the memory bandwidth initially improves task accuracy but results in diminishing returns at larger scales. With 512 memory banks (approximately $0.2M$ parameters), the memory achieves near-perfect accuracy (0.99), demonstrating sufficient capacity for basic operations. However, when scaling to 1024 memory banks (about $0.4M$ parameters), accuracy drops to 0.98, likely due to increased noise from sparsely activated memory units. *Generality*, which reflects the memory's ability to generalize to semantically equivalent queries, shows a similar trend to accuracy, further supporting this observation. Based on these results, we adopt 512 memory banks as the default memory bandwidth in this paper. For most applications, we recommend dynamically allocating bandwidth based on task-specific requirements, optimizing both performance and efficiency.

4.3.2. Memory Depth

Memory depth controls the size of each memory bank, with larger depth allowing for richer information refinement but also increasing computational costs. As illustrated in the

Table 4. ESD of different memories on Llama3 (8B).

Memory	GRACE (*W.M.*)		WISE (*I.M.*)		Ours (*P.M.*)	
#Data	1K	10K	1K	10K	1K	10K
Accuracy↑	0.0794	0.0651	0.5512	0.2720	**0.9908**	**0.9878**
#Param↓	18.4M	184.3M	234.9M	234.9M	**0.2M**	**0.2M**
ESD↑	0.0043	0.0035	0.0023	0.0116	**4.95**	**49.39**

Table 5. ESD of different memories across LLMs ($3K$ data).

Memory	GRACE (*W.M.*)		WISE (*I.M.*)		Ours (*P.M.*)	
LLMs	GPTJ	Llama3	GPTJ	Llama3	GPTJ	Llama3
Accuracy↑	0.3368	0.0624	0.4598	0.3348	**0.9985**	**0.9805**
#Param↓	61.4M	55.3M	268.4M	234.9M	**0.2M**	**0.2M**
ESD↑	0.0165	0.0034	0.0051	0.0043	**14.98**	**14.71**

bottom subfigure of Figure 5, both the storage *Accuracy* and *Generality* improve as the memory depth increases. Model performance stabilizes when the depth reaches 11, indicating that larger depth enhances the memory's storage capacity. However, this improvement comes at the cost of higher parameter counts in the memory, leading to increased computational overhead during memory writing and retrieval operations. To strike a balance between performance and efficiency, we choose a memory depth of 11 as an optimal configuration for our framework.

4.4. Discussion

4.4.1. Memory Storage Capacity

As discussed in Section 4.3, we configured the memory with a bandwidth of 512 and depth of 11. This setup raises the question: *What is the storage capacity of the peripheral memory*, similar to the storage size of a physical RAM. To investigate this, we continuously stored data in memory while evaluating its performance and storage efficiency. As shown in Figure 6, as the amount of stored data increases, the memory storage accuracy remains high (around 98% at $10K$). However, the generalization ability for semantically equivalent queries significantly drops, from 0.72 at $0.5K$ to 0.65 at $10K$, especially as the data volume reaches 1000. This highlights the trade-off between memory utilization and semantic discriminability, which arises from the direct querying mechanism using the last token hidden state.

4.4.2. Effective Storage Density (ESD)

Memory storage density is a crucial factor for evaluating the memory efficiency. Similar to knowledge capacity (Allen-Zhu & Li, 2024), we introduce the concept of *Effective Storage Density* (**ESD**) to quantify how many pieces of knowledge are successfully stored per $1K$ parameters:

$$\textbf{ESD}(\mathcal{M}|\mathcal{D}) = \frac{\#\mathcal{D}}{\#\Theta(\mathcal{M})} * \text{Accuracy}(\mathcal{D}|\mathcal{M}) \qquad (9)$$

where $\#\mathcal{D}$ refers to the number of data in $\mathcal{D}$. $\#\Theta(\mathcal{M})$ calculates the parameters number of $\mathcal{M}$ in KB. Accuracy$(\cdot)$ estimates the storage accuracy of $\mathcal{D}$ based on $\mathcal{M}$.

Table 4 compares the effective storage density across different memory types. For working memory[5], which caches contextual key-value pairs, the amount of stored information increases dramatically as the data volume grows. This leads to a significant rise in the number of parameters required for memory (*e.g.,* increasing from $18.4M$ at $1K$ to $184.3M$ at $10K$). However, as the cached information increases, the probability of retrieving irrelevant data also rises during inference, reducing storage performance (*e.g.,* reducing from 0.0794 at $1K$ to 0.0651 at $10K$). These two factors cause a substantial decrease in effective storage density (*e.g.,* from 0.0043 at $1K$ to 0.0035 at $10K$). In implicit memory[6], the number of parameters remains relatively stable as the data increases on the same LLM, but its performance significantly drops (*e.g.,* from 0.5512 at $1K$ to 0.2720 at $10K$) due to knowledge forgetting, causing poor ESD. Additionally, the number of parameters in working memory and implicit memory inevitably changes as the model changes (as shown in Table 5), which will bring inconvenience for memory management. At the same time, both of them rely heavily on the LLMs' structure, limiting their configurability.

In contrast, the proposed peripheral memory shows stable performance across different LLMs and varying data sizes. Moreover, by leveraging KANs, peripheral memory significantly reduces the number of parameters required (only $0.2M$). This enables it to achieve considerably higher effective storage density, *e.g.,* 1000^+ times greater than working memory and 500^+ times greater than implicit memory for $1K$ data under Llama3 (8B). Surprisingly, this advantage grows even more pronounced as the data size increases or as the LLM architecture changes (See Table 4 and Table 5).

5. Related Work

Existing memory mechanisms for LLMs can be broadly classified into three categories (Yang et al., 2024): working memory, implicit memory, and explicit memory. Working Memory retains transient context by caching contextual key-value pairs from hidden layers. Notable methods include Transformer-XL (Dai et al., 2019), which introduces segment-level recurrence for cross-context retention, and Memformer (Wu et al., 2022a), which dynamically updates

[5]#Parameters are calculated based on Hartvigsen et al. (2023).
[6]#Parameters are calculated based on Wang et al. (2024a).

memory slots during decoding. Subsequent advancements enhance working memory through methods like approximate kNN lookup (Wu et al., 2022b), subject word embedding alteration (Li et al., 2025), retrieval-based methods (Hartvigsen et al., 2023), and retrieval-augmented counterfactual models (Mitchell et al., 2022b). However, these approaches focus on short-term context and lack persistent, scalable storage. Implicit memory encodes knowledge into model parameters for long-term retention. Approaches in this category include fine-tuning (Hu et al., 2022; Modarressi et al., 2025; Yu et al., 2024), direct parameter edits (Mitchell et al., 2022a; Meng et al., 2023; Raunak & Menezes, 2022)), and parameter-efficient patching (Dai et al., 2023; Wang et al., 2024b; Wang & Li, 2024; Wang et al., 2024a). While effective for targeted updates, these methods often suffer from catastrophic forgetting and architectural rigidity, as they are tightly coupled with the LLM parameters (Kaddour et al., 2023). Explicit memory encompasses retrievable model parameters, externalized knowledge, or sparsely-activated neural circuits (Yang et al., 2024). However, these frameworks often rely on static memory allocation (Yao et al., 2023), struggling with issues related to scalability, reusability and configurability.

6. Conclusion

In this paper, we introduced peripheral memory, a novel memory augmentation framework inspired by the decoupled architecture of physical RAM and CPU. By conceptualizing memory as an independent module separate from the LLM, our method addressed critical limitations in scalability, reusability, and configurability inherent to existing memories. At the core, the Kolmogorov-Arnold Networks were used to model memory banks, enabling smooth state transitions and sequential interactions between storage units. Additionally, a dedicated confidence mechanism further ensured reliable retrieval by filtering irrelevant or noisy data, preserving the LLM's original knowledge integrity. Extensive experiments on knowledge editing demonstrated its superior performance while maintaining efficiency.

Limitations

In the proposed framework, memory is queried directly using the hidden state of the last input token. While this design facilitates efficient retrieval, it faces a well-known challenge in representation learning: hypersensitivity to minor input variations (Jiang et al., 2020). Semantically equivalent queries (despite conveying the same meaning) may be projected to distinct regions of the memory space due to subtle differences in token-level hidden representations. Moreover, as the volume of stored data increases, the memory module tends to overfit to the original query distribution, becoming increasingly specialized in retrieving exact matches for frequently encountered hidden states. Consequently, this specialization comes at the cost of reduced generalization to semantically similar but representationally divergent queries (see Figure 6 and Section 4.4.1).

To overcome limitations of *the direct memory querying*, we are now actively developing a memory management module inspired by the Memory Management Unit (MMU) in operating systems. This module serves as an abstraction module between the LLM and the peripheral memory, effectively *decoupling semantic alignment from storage operations*. By doing so, the peripheral memory can specialize in storage and retrieval, while the MMU is responsible for query normalization and semantic mapping.

Acknowledgments

This work is partially supported by National Nature Science Foundation of China under No. 62476058. We thank the Big Data Computing Center of Southeast University for providing the facility support on the numerical calculations in this paper.

Impact Statement

This paper presents work that aims to advance the field of memory-augmented LLMs, with the potential to significantly advance the field of machine learning. There are many potential consequences of our work:

Societal Implications: Peripheral memory has the potential to democratize access to LLM capabilities by enabling efficient, modular memory systems. However, its deployment must be accompanied by rigorous ethical guidelines and regulatory oversight to ensure responsible use.

Ethical Impacts: The modular design of peripheral memory enhances interpretability, as memory operations are decoupled from the LLM's core processing. This transparency is crucial for auditing and debugging memory systems.

Environmental Impact: The reusability and read&write efficiency of peripheral memory significantly reduce redundant storage of identical knowledge, enhancing the overall energy efficiency of the memory system. By minimizing repetitive operations, our framework lowers the computational resources required for memory management, thereby reducing the carbon footprint of memory-augmented LLMs. This aligns with broader sustainability goals in AI research, promoting environmentally responsible development of advanced language models.

Potential Risks: Storing and retrieving dynamic knowledge raises privacy risks, particularly if sensitive or personal data is inadvertently memorized. Techniques like differential privacy or memory anonymization should be investigated.

References

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.1, knowledge storage and extraction. In *ICML*. OpenReview.net, 2024. URL https://openreview.net/forum?id=5x788rqbcj.

Azaria, A., Azoulay, R., and Reches, S. Chatgpt is a remarkable tool—for experts. *Data Intelligence*, 6:240–296, 2024. URL http://www.sciengine.com/doi/10.1162/dint_a_00235.

Aziznejad, S. and Unser, M. Deep spline networks with control of lipschitz regularity. In *IEEE ICASSP*, pp. 3242–3246, 2019. URL https://ieeexplore.ieee.org/document/8682547.

Blum, B. I. *History of Computers*, pp. 3–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986. ISBN 978-3-662-26537-6. doi: 10.1007/978-3-662-26537-6_1. URL https://doi.org/10.1007/978-3-662-26537-6_1.

Bohra, P., Campos, J., Gupta, H., Aziznejad, S., and Unser, M. Learning activation functions in deep (spline) neural networks. *IEEE Open Journal of Signal Processing*, 1: 295–309, 2020. doi: 10.1109/OJSP.2020.3039379.

Chen, H. Large knowledge model: Perspectives and challenges. *Data Intelligence*, 6:587–620, 2024. URL http://www.sciengine.com/doi/10.3724/2096-7004.di.2024.0001.

Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., and Wei, F. Knowledge neurons in pretrained transformers. In *ACL*, pp. 8493–8502. Association for Computational Linguistics, 2022. URL https://doi.org/10.18653/v1/2022.acl-long.581.

Dai, D., Jiang, W., Dong, Q., Lyu, Y., and Sui, Z. Neural knowledge bank for pretrained transformers. In *NLPCC*, volume 14303 of *Lecture Notes in Computer Science*, pp. 772–783. Springer, 2023. URL https://doi.org/10.1007/978-3-031-44696-2_60.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*, pp. 2978–2988. Association for Computational Linguistics, 2019. URL https://aclanthology.org/P19-1285/.

Fang, J., Jiang, H., Wang, K., Ma, Y., Wang, X., He, X., and Chua, T. Alphaedit: Null-space constrained knowledge editing for language models. In *ICLR*, 2025. URL https://openreview.net/forum?id=HvSytvg3Jh.

Guo, X., Pan, J., Wang, X., Chen, B., Jiang, J., and Long, M. On the embedding collapse when scaling up recommendation models. In *ICML*. JMLR.org, 2024.

Gupta, A., Baskaran, S., and Anumanchipalli, G. Rebuilding ROME : Resolving model collapse during sequential model editing. In *EMNLP*, pp. 21738–21744. Association for Computational Linguistics, 2024a. URL https://aclanthology.org/2024.emnlp-main.1210.

Gupta, A., Sajnani, D., and Anumanchipalli, G. A unified framework for model editing. In Al-Onaizan, Y., Bansal, M., and Chen, Y. (eds.), *EMNLP*, pp. 15403–15418. Association for Computational Linguistics, 2024b. URL https://aclanthology.org/2024.findings-emnlp.903.

Hartvigsen, T., Sankaranarayanan, S., Palangi, H., Kim, Y., and Ghassemi, M. Aging with GRACE: lifelong model editing with discrete key-value adaptors. In *NeurIPS*, volume 36, pp. 47934–47959. Curran Associates, Inc., 2023. URL https://dl.acm.org/doi/10.5555/3666122.3668201.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-rank adaptation of large language models. In *ICLR*, Virtual, 2022. OpenReview.net. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jacob, B., Ng, S., and Wang, D. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 0123797519.

Jiang, W. X., Nelson, B. L., and Hong, L. J. Estimating sensitivity to input model variance. In *WSC*, pp. 3705—-3716. IEEE Press, 2020. ISBN 9781728132839. URL https://informs-sim.org/wsc19papers/395.pdf.

Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., and McHardy, R. Challenges and applications of large language models. *ArXiv*, abs/2307.10169, 2023. URL https://api.semanticscholar.org/CorpusID:259982665.

Kolmogorov, A. N. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk*, 114:953–956, 1957. URL http://mi.mathnet.ru/dan22050.

Levy, O., Seo, M., Choi, E., and Zettlemoyer, L. Zero-shot relation extraction via reading comprehension. In Levy, R. and Specia, L. (eds.), *CoNLL*, pp. 333–342. Association for Computational Linguistics, 2017. URL https://doi.org/10.18653/v1/K17-1034.

Li, X., Li, S., Song, S., Yang, J., Ma, J., and Yu, J. PMET: precise model editing in a transformer. In *AAAI*, pp. 18564–18572. AAAI Press, 2024. URL https://doi.org/10.1609/aaai.v38i17.29818.

Li, X., Li, S., Song, S., Liu, H., Ji, B., Wang, X., Ma, J., Yu, J., Liu, X., Wang, J., and Zhang, W. Swea: Updating factual knowledge in large language models via subject word embedding altering. *AAAI*, 39 (23):24494–24502, 2025. URL https://ojs.aaai.org/index.php/AAAI/article/view/34628.

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T. Y., and Tegmark, M. Kan: Kolmogorov-arnold networks, 2024. URL https://arxiv.org/abs/2404.19756.

Llama Team, A. . M. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. URL https://github.com/meta-llama/llama3/.

Luo, M., Xu, X., Dai, Z., Pasupat, P., Kazemi, M., Baral, C., Imbrasaite, V., and Zhao, V. Y. Dr.icl: Demonstration-retrieved in-context learning. *Data Intelligence*, 6:909–922, 2024. URL http://www.sciengine.com/doi/10.3724/2096-7004.di.2024.0012.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in GPT. In *NeurIPS*, pp. 17359–17372. Curran Associates, Inc., 2022. URL https://rome.baulab.info/.

Meng, K., Sharma, A. S., Andonian, A. J., Belinkov, Y., and Bau, D. Mass-editing memory in a transformer. In *ICLR*, Virtual, 2023. OpenReview.net. URL https://openreview.net/pdf?id=MkbcAHIYgyS.

Mitchell, E., Lin, C., Bosselut, A., Finn, C., and Manning, C. D. Fast model editing at scale. In *ICLR*, Virtual, 2022a. OpenReview.net. URL https://openreview.net/forum?id=0DcZxeWfOPt.

Mitchell, E., Lin, C., Bosselut, A., Manning, C. D., and Finn, C. Memory-based model editing at scale. In *ICML*, volume 162, pp. 15817–15831. PMLR, 2022b. URL https://proceedings.mlr.press/v162/mitchell22a.html.

Modarressi, A., Köksal, A., Imani, A., Fayyaz, M., and Schuetze, H. MemLLM: Finetuning LLMs to use explicit read-write memory. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=dghM7sOudh.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Barnes, N., and Mian, A. A comprehensive overview of large language models. *CoRR*, abs/2307.06435, 2024. URL https://arxiv.org/pdf/2307.06435.

Raunak, V. and Menezes, A. Rank-one editing of encoder-decoder models. In *The Second Workshop on InterNLP of NeurIPS*, 2022. URL https://doi.org/10.48550/arXiv.2211.13317.

Rojas, R. and Hashagen, U. (eds.). *The first computers: history and architectures*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0262181975.

Singh, A. Exploring language models: A comprehensive survey and analysis. In *RMKMATE*, pp. 1–4, 2023. doi: 10.1109/RMKMATE59243.2023.10369423.

Wang, B. and Komatsuzaki, A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, 2021.

Wang, P., Li, Z., Zhang, N., Xu, Z., Yao, Y., Jiang, Y., Xie, P., Huang, F., and Chen, H. WISE: Rethinking the knowledge memory for lifelong model editing of large language models. In *NeurIPS*, 2024a. URL https://openreview.net/forum?id=VJMYOfJVC2.

Wang, R. and Li, P. Memoe: Enhancing model editing with mixture of experts adaptors. *CoRR*, abs/2405.19086, 2024. URL https://doi.org/10.48550/arXiv.2405.19086.

Wang, Y., Gao, Y., Chen, X., Jiang, H., Li, S., Yang, J., Yin, Q., Li, Z., Li, X., Yin, B., Shang, J., and McAuley, J. J. MEMORYLLM: towards self-updatable large language models. In *ICML*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=p0lKWzdikQ.

Wu, Q., Lan, Z., Qian, K., Gu, J., Geramifard, A., and Yu, Z. Memformer: A memory-augmented transformer for sequence modeling. In *Findings of AACL-IJCNLP*, pp. 308–318. Association for Computational Linguistics, 2022a. URL https://aclanthology.org/2022.findings-aacl.29/.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. In *ICLR*. OpenReview.net, 2022b. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Yang, H., Lin, Z., Wang, W., Wu, H., Li, Z., Tang, B., Wei, W., Wang, J., Tang, Z., Song, S., Xi, C., Yu, Y., Chen, K., Xiong, F., Tang, L., and E, W. Memory^3: Language modeling with explicit memory. *Journal of Machine Learning*, 3(3):300–346, 2024. URL http://global-sci.org/intro/article_detail/jml/23419.html.

Yao, Y., Wang, P., Tian, B., Cheng, S., Li, Z., Deng, S., Chen, H., and Zhang, N. Editing large language models: Problems, methods, and opportunities. In *EMNLP*, pp.

10222–10240. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.632/.

Yu, L., Chen, Q., Zhou, J., and He, L. MELO: enhancing model editing with neuron-indexed dynamic lora. In *AAAI*, pp. 19449–19457. AAAI Press, 2024. URL https://doi.org/10.1609/aaai.v38i17.29916.

Zhang, N., Yao, Y., Tian, B., Wang, P., Deng, S., Wang, M., Xi, Z., Mao, S., Zhang, J., Ni, Y., Cheng, S., Xu, Z., Xu, X., Gu, J., Jiang, Y., Xie, P., Huang, F., Liang, L., Zhang, Z., Zhu, X., Zhou, J., and Chen, H. A comprehensive study of knowledge editing for large language models. *CoRR*, abs/2401.01286, 2024. URL https://doi.org/10.48550/arXiv.2401.01286.

Zheng, C., Li, L., Dong, Q., Fan, Y., Wu, Z., Xu, J., and Chang, B. Can we edit factual knowledge by in-context learning? In *EMNLP*, pp. 4862–4876. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.296.

Zhu, C., Rawat, A. S., Zaheer, M., Bhojanapalli, S., Li, D., Yu, F. X., and Kumar, S. Modifying memories in transformer models. *CoRR*, abs/2012.00363, 2020. URL https://arxiv.org/abs/2012.00363.