Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

Memory^3: Language Modeling with Explicit Memory

Hongkang Yang * [1], Zehao Lin[1], Wenjin Wang[1], Hao Wu[1], Zhiyu Li[1], Bo Tang[1], Wenqiang Wei[1], Jinbo Wang[1,4], Zeyun Tang[1], Shichao Song[1], Chenyang Xi[1], Yu Yu[1], Kai Chen[1], Feiyu Xiong[1], Linpeng Tang[2], and Weinan E[1,3,4,5]

[1]Center for LLM, Institute for Advanced Algorithms Research, Shanghai 200233, China.
[2]Moqi Inc, Beijing 100080, China.
[3]Center for Machine Learning Research, Peking University, Beijing 100871, China.
[4]School of Mathematical Sciences, Peking University, Beijing 100871, China.
[5]AI for Science Institute, Beijing 100083, China.

Abstract. The training and inference of large language models (LLMs) are together a costly process that transports knowledge from raw data to meaningful computation. Inspired by the memory hierarchy of the human brain, we reduce this cost by equipping LLMs with explicit memory, a memory format cheaper than model parameters and text retrieval-augmented generation (RAG). Conceptually, with most of its knowledge externalized to explicit memories, the LLM can enjoy a smaller parameter size, training cost, and inference cost, all proportional to the amount of remaining "abstract knowledge". As a preliminary proof of concept, we train from scratch a 2.4 B LLM, which achieves better performance than much larger LLMs as well as RAG models, and maintains higher decoding speed than RAG. The model is named Memory^3, since explicit memory is the third form of memory in LLMs after implicit memory (model parameters) and working memory (context key-values). We introduce a memory circuitry theory to support the externalization of knowledge, and present novel techniques including a memory sparsification mechanism that makes storage tractable and a two-stage pretraining scheme that facilitates memory formation.

Keywords:
Large language model,
Explicit memory,
Large-scale pretraining,
Efficient inference,
AI database.

Article Info.:
Volume: 3
Number: 3
Pages: 300 - 346
Date: September/2024
doi.org/10.4208/jml.240708

Article History:
Received: 08/07/2024
Accepted: 20/08/2024

Communicated by:
Zhi-Qin Xu

1 Introduction

Large language models have enjoyed unprecedented popularity in recent years thanks to their extraordinary performance [1,2,6,9,51,53,108,125]. The prospect of scaling laws [50, 57, 95] and emergent abilities [101, 117] constantly drives for substantially larger models, resulting in the rapid increase in the cost of LLM training and inference. People have been trying to reduce this cost through optimizations in various aspects, including architecture [3,27,37,71,86,107], data quality [45,55,63,100], operator [29,60], parallelization [59,88,92, 98], optimizer [67,115,123], scaling laws [50,126], generalization theory [52,130], hardware [30], etc.

We introduce the novel approach of optimizing knowledge storage. The combined cost of LLM training and inference can be seen as the cost of encoding the knowledge from text

*Corresponding author: hongkang@alumni.princeton.edu

data into various memory formats, plus the cost of reading from these memories during inference

$$\sum_{\text{knowledge } k} \min_{\text{format } m} \text{cost}_{\text{write}}(k, m) + n_k \cdot \text{cost}_{\text{read}}(k, m), \tag{1.1}$$

where $\text{cost}_{\text{write}}$ is the cost of encoding a piece of knowledge k into memory format m, $\text{cost}_{\text{read}}$ is the cost of integrating k from format m into inference, and n_k is the expected usage count of this knowledge during the lifespan of this LLM (e.g. a few months for each version of ChatGPT [8,83]). The definitions of knowledge and memory in the context of LLMs are provided in Section 2, and this paper uses knowledge as a countable noun. Typical memory formats include model parameters and plain text for retrieval-augmented generative models, their write functions and read functions are listed in Table 1.1, and their $\text{cost}_{\text{write}}$ and $\text{cost}_{\text{read}}$ are provided in Fig. 1.1.

We introduce a new memory format, explicit memory, characterized by moderately low write cost and read cost. As depicted in Fig. 1.2, our model first converts a knowledge base (or any text dataset) into explicit memories, implemented as sparse attention key-values, and then during inference, recalls these memories and integrates them into the self-

Table 1.1: Analogy of the memory hierarchies of humans and LLMs.

Memory format of humans	Example	Memory format of LLMs	Write	Read
Implicit memory	Common expressions	Model parameters	Training	Matrix multiplication
Explicit memory	Books read	This work	Memory encoding	Self-attention
External information	Open-book exam	Plain text (RAG)	None	Encode from scratch

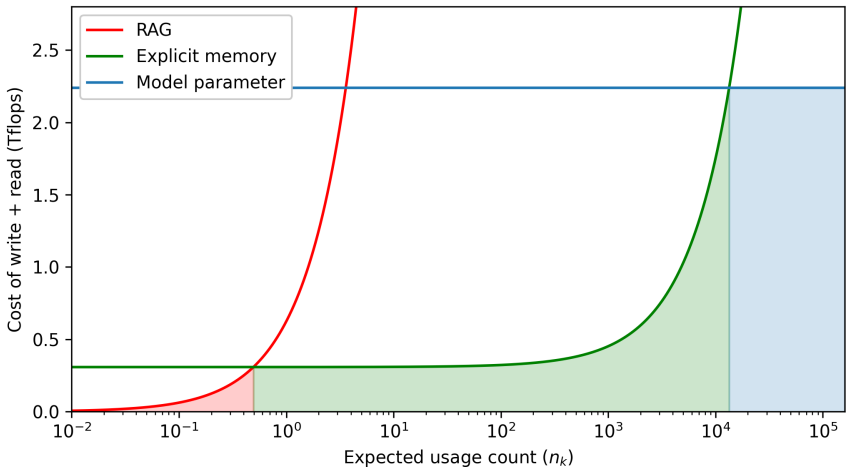

Figure 1.1: The total cost (TFlops) of writing and reading a piece of knowledge by our 2.4 B model with respect to its expected usage count. The curves represent the cost of different memory formats, and the shaded area represents the minimum cost given the optimal format. The plot indicates that $(0.494, 13400)$ is the advantage interval for explicit memory. The calculations are provided in Appendix A. (The blue curve is only a coarse lower bound on the cost of model parameters.)

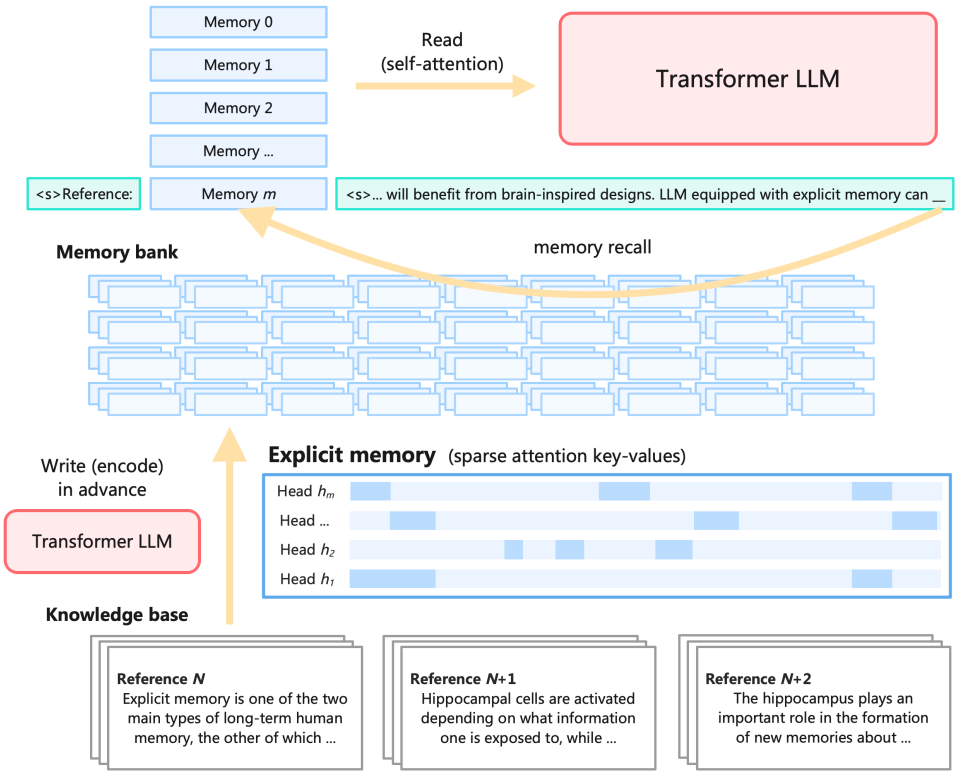

Figure 1.2: The Memory^3 model converts texts to explicit memories, and then recalls these memories during inference. The explicit memories can be seen as retrievable model parameters, externalized knowledge, or sparsely-activated neural circuits.

attention layers. Our design is simple so that most of the existing Transformer-based LLMs should be able to accommodate explicit memories with a little finetuning, and thus it is a general-purpose "model amplifier". Eventually, it should reduce the cost of pretraining LLMs, since there will be much less knowledge that must be stored in parameters, and thus less training data and smaller model size.

The new memory format enables us to define a memory hierarchy for LLMs:

$$\text{plain text (RAG)} \ \rightarrow \ \text{explicit memory} \ \rightarrow \ \text{model parameter}$$

such that by going up the hierarchy, $\text{cost}_{\text{write}}$ increases while $\text{cost}_{\text{read}}$ decreases. To minimize the cost (1.1), one should store each piece of knowledge that is very frequently/rarely used in the top/bottom of this hierarchy, and everything in between as explicit memory. As illustrated in Table 1.1, the memory hierarchy of LLMs closely resembles that of humans. For humans, the explicit/implicit memories are the long-term memories that are acquired and used consciously/unconsciously [56].

As a remark, one can compare the plain LLMs to patients with impaired explicit memory, e.g. due to injury to the medial temporal lobe. These patients are largely unable to learn semantic knowledge (usually stored as explicit memory), but can acquire sensori-

motor skills through repetitive priming (stored as implicit memories) [10, 23, 39]. Thus, one may hypothesize that due to the lack of explicit memory, the training of plain LLMs is as inefficient as repetitive priming, and thus has ample room for improvement. In analogy with humans, for instance, it is easy to recall and talk about a book we just read, but to recite it as unconsciously as tying shoe laces requires an enormous effort to force this knowledge into our muscle memory. From this perspective, it is not surprising that LLM training consumes so much data and energy [73, 120]. We want to rescue LLMs from this poor condition by equipping it with an explicit memory mechanism as efficient as that of humans.

A quantitative illustration of the cost (1.1) is given by Fig. 1.1, where we characterize $\text{cost}_{\text{write}}$ and $\text{cost}_{\text{read}}$ by the amount of compute (TFlops). The plot indicates that if a piece of knowledge has an expected usage count $\in (0.494, 13400)$, then it is optimal to be stored as an explicit memory. Moreover, the introduction of explicit memory helps to externalize the knowledge stored in model parameters and thus allow us to use a lighter backbone, which ultimately reduces all the costs in Fig. 1.1.

The second motivation for explicit memory is to alleviate the issue of knowledge traversal. Knowledge traversal happens when the LLM wastefully invokes all its parameters (and thus all its knowledge) each time it generates a token. As an analogy, it is unreasonable for humans to recall everything they learned whenever they write a word. Let us define the knowledge efficiency of an LLM as the ratio of the minimum amount of knowledge sufficient for one decoding step to the amount of knowledge actually used. An optimistic estimation of knowledge efficiency for a 10 B LLM is 10^{-5}: On one hand, it is unlikely that generating one token would require more than 10^4 bits of knowledge (roughly equivalent to a thousand-token long passage, sufficient for enumerating all necessary knowledge); on the other hand, each parameter is involved in the computation and each stores at least 0.1 bit of knowledge [4, Result 10] (this density could be much higher if the LLM is trained on cleaner data), thus using 10^9 bits in total.

A novel architecture is needed to boost the knowledge efficiency of LLMs from 10^{-5} to 1, whereas current designs are far from this goal. Consider the mixture-of-experts architecture (MoE) for instance, which uses multiple MLP layers (experts) in each Transformer block and process each token with only a few MLPs. The boost of MoE, namely the ratio of the total amount of parameters to the amount of active parameters, is usually bounded by $4 \sim 32$ [37, 53, 99]. Similarly, neither the mixture-of-depth architecture [34, 91] nor sparsified MLP neurons and attention heads [71] can bring greater gains. RAG appears very sparse if we compare the amount of retrieved texts with the size of the text database; nevertheless, RAG is usually built upon a plain LLM as backbone, which provides most of the knowledge used in inference, and thus offers little assistance in addressing the knowledge traversal problem.

An ideal solution is to retrieve only the needed parameters for each token. This is naturally achieved by explicit memories if we compare memory recall to parameter retrieval.

The third motivation is that, as a human-like design, explicit memory enables LLMs to develop more human-like capabilities. To name a few:

- Infinitely long context: LLMs have the difficulty of processing long texts since their working memory (context key-values) costs too much GPU memory and compute.

Meanwhile, despite that humans have very limited working memory capacity [24, 25], they can manage to read and write long texts by converting working memories to explicit memories (thus saving space) and retrieving only the needed explicit memories for inference (thus saving compute). Similarly, by saving explicit memories on drives and doing frequent and constant-size retrieval, LLMs can handle arbitrarily long contexts with time complexity $\mathcal{O}(l \log l)$ instead of $\Theta(l^2)$, where l is the context length.

- Memory consolidation: Instead of writing a piece of knowledge directly into implicit memory, i.e. training model parameters, LLM can first convert it to explicit memory through plain encoding, and then convert this explicit memory to implicit memory through a low-cost step such as compression and finetuning, thus reducing the overall cost.
- Factuality and interpretability: Encoding texts as explicit memories is less susceptible to information loss compared to dissolving them in model parameters. With more factual details provided by explicit memories, the LLMs would have less tendency to hallucinate. Meanwhile, the correspondence of explicit memories to readable texts makes the inference more transparent to humans, and also allows the LLM to consciously examine its own thought process.

We demonstrate the improved factuality in the experiments section, and leave the rest to future work.

In this work, we introduce a novel architecture and training scheme for LLM based on explicit memory. The architecture is called Memory^3, as explicit memory is the third form of memory in LLM after working memory (context key-values) and implicit memory (model parameters).

- Memory^3 utilizes explicit memories during inference, alleviating the burden of model parameters to memorize specific knowledge.
- The explicit memories are encoded from our knowledge base, and our sparse memory format maintains a realistic storage size.
- We trained from scratch a Memory^3 model with 2.4 B non-embedding parameters, and its performance surpasses SOTA models with greater sizes. It also enjoys better performance and faster inference than RAG. These results are illustrated in Fig. 1.3.
- Furthermore, Memory^3 boosts factuality and alleviates hallucination, and it enables fast adaptation to professional tasks.

This paper is structured as follows: Section 2 lays the theoretical foundation for Memory^3, in particular our definitions of knowledge and memory. Section 3 discusses the basic design of Memory^3, including its architecture and training scheme. Sections 4, 5, and 6 describes the training of Memory^3. Section 7 evaluates the performance of Memory^3 on general benchmarks and professional tasks. Finally, Section 8 concludes this paper and discusses future works.

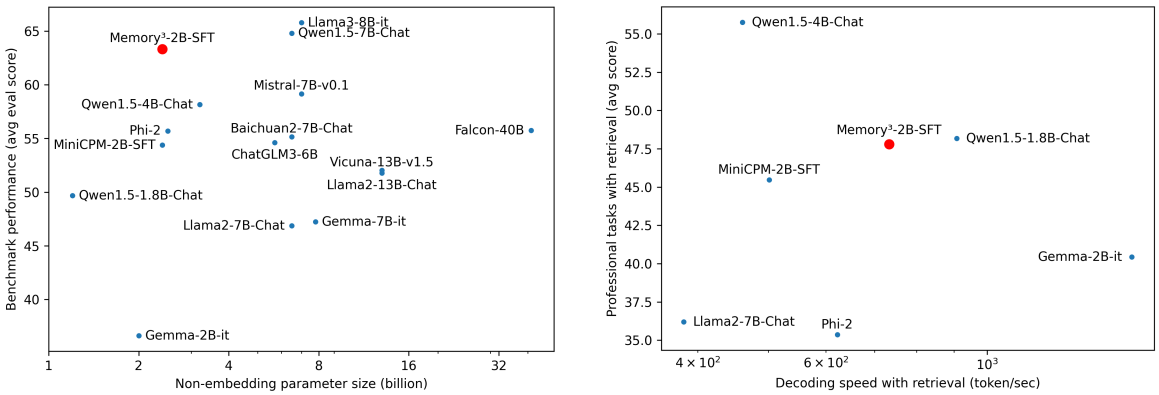

Figure 1.3: Left: Performance on benchmarks, with respect to model size (top-left is better). Right: Retrieval-augmented performance on professional tasks, versus decoding speed with retrieval (top-right is better). The left plot is based on Table 7.1. The right plot is based on Tables 7.5 and 7.6. Memory^3 uses high frequency retrieval of explicit memories, while the RAG models use a fixed amount of 5 references. This is a preliminary experiment and we have not optimized the quality of our pretraining data as well as the efficiency of our inference pipeline, so the results may not be comparable to those of the SOTA models.

1.1 Related work

1.1.1 Retrieval-augmented training

Several language models have incorporated text retrieval from the pretraining stage. REALM [46] augments a BERT model with one retrieval step to solve QA tasks. Retro [14] enhances auto-regressive decoding with multiple rounds of retrieval, once per 64 tokens. The retrieved texts are injected through a two-layer encoder and then several cross-attention layers in the decoder. Retro++ [111] explores the scalability of Retro by reproducing Retro up to 9.5 B parameters.

Meanwhile, several models are adapted to retrieval in the finetuning stage. WebGPT [80] learns to use search engine through imitation learning in a text-based web-browsing environment. Toolformer [96] performs decoding with multiple tools including search engine, and the finetuning data is labeled by the LM itself.

The closest model to ours is Retro. Unlike explicit memory, Retro needs to encode the retrieved texts in real-time during inference. To alleviate the cost of encoding these references, it chooses to use a separate, shallow encoder and also retrieve few references. Intuitively, this compromise greatly reduces the amount of knowledge that can be extracted and supplied to inference.

Another line of research utilizes retrieval to aid long-context modeling. Memorizing Transformer [122] extends the context of language models by an approximate kNN lookup into a non-differentiable cache of past key-value pairs. LongLlama [110] enhances the discernability of context key-value pairs by a finetuning process inspired by contrastive learning. LONGMEM [116] designs a decoupled architecture to avoid the memory staleness issue when training the Memorizing Transformer. These methods are not directly applicable to large knowledge bases since the resulting key-value caches will occupy enor-

mous space. Our method overcomes this difficulty through a more intense memory sparsification method.

1.1.2 Sparse computation

To combat the aforementioned knowledge traversal problem and improve knowledge efficiency, ongoing works seek novel architectures that process each token with a minimum and adaptive subset of model parameters. This adaptive sparsity is also known as contextual sparsity [71]. The Mixture-of-Experts use sparse routing to assign Transformer submodules to tokens, scaling model capacity without large increases in training or inference costs. The most common MoE design [37] hosts multiple MLP layers in each Transformer block and routes each token to a few MLPs with the highest allocation score predicted by a linear classifier. Furthermore, variants based on compression such as QMoE [38] are introduced to alleviate the memory burden of MoE. Despite the sparse routing, the boost in parameter efficiency is usually bounded by $4 \sim 32$. For instance, the Arctic model [99], one of the sparsest MoE LLM in recent years, has an active parameter ratio of about 3.5%. Similarly, the Mixture of Depth architecture processes each token with an adaptive subset of the model layers. The implementations can be based on early exit [34] or top-k routing [91], reducing the amount of compute to $12.5 \sim 50\%$. More fine-grained approaches can perform sparsification at the level of individual MLP neurons and attention heads. The model Deja Vu [71] trains a low-cost network for each MLP/attention layer that predicts the relevance of each neuron/head at this layer to each token. Then, during inference, Deja Vu keeps the top $5 \sim 15\%$ MLP neurons and $20 \sim 50\%$ attention heads for each token.

1.1.3 Parameter as memory

Several works have portrayed model parameters as implicit memory, in accordance with our philosophy. [43] demonstrates that the neurons in the MLP layers of GPTs behave like key-value pairs. Specifically, with the MLP layer written as $\sigma(XK^\top)V$, each row of the first layer weight K_i functions like a key vector, with the corresponding row in the second layer weight V_i being the value vector. [43] observes that for most of the MLP neurons, the K_i is activated by context texts that obey some human interpretable pattern, and the V_i activates the column of the output matrix that corresponds to the most probable next token of the pattern (e.g. n-gram). Based on this observation, [104] designs a GPT variant that consists of only attention layers, with performance matching that of the usual GPTs. The MLP layers are incorporated into the attention layers in the form of key-value vector pairs, which are called persistent memories. Similarly, using sensitivity analysis, [26] discovers that factual knowledge learned by BERT is often localized at one or few MLP neurons. These neurons are called "knowledge neurons", and by manipulating them, [26] manages to update single pieces of knowledge of BERT. Meanwhile, [35] studies an interesting phenomenon known as superposition or polysemanticity, that a neural network can store many unrelated concepts into a single neuron.

2 Memory circuitry theory

This section introduces our memory circuitry theory, which defines knowledge and memory in the context of LLM. We will see that this theory helps to determine which knowledge can be stored as explicit memory, and what kind of model architecture is suitable for reading and writing explicit memories. For readers interested primarily in the results, it may suffice to review Claim 2.1 and Remark 2.1 before proceeding to the subsequent sections. The concepts to be discussed are illustrated in Fig. 2.1.

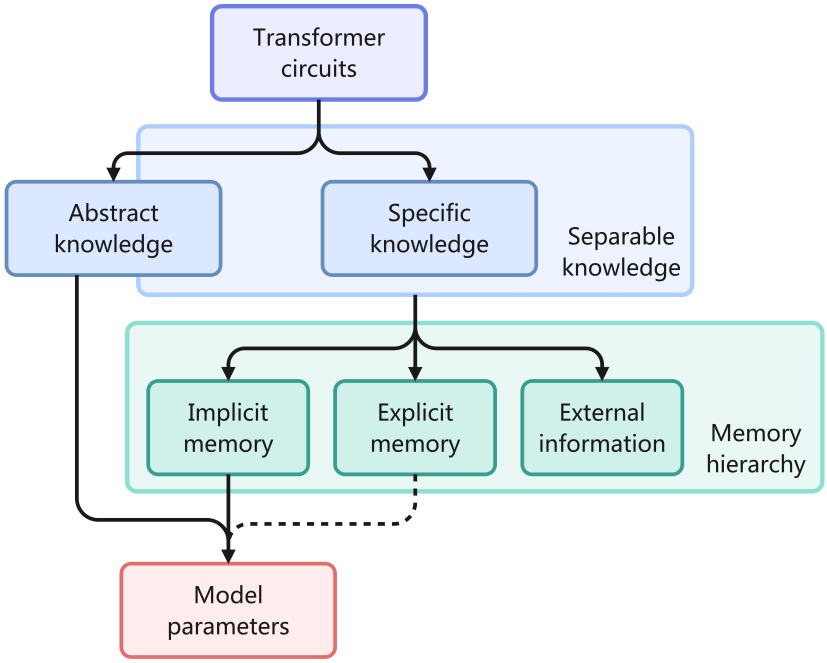

Figure 2.1: Categorization of knowledge and memory formats. The explicit memories, extracted from model activations, lie half-way between raw data and model parameters, so we use a dotted line to indicate that they may or may not be regarded as parameters.

2.1 Preliminaries

The objective is to decompose the computations of a LLM into smaller, recurring parts, and analyze which parts can be separated from the LLM. These small parts will be defined as the "knowledge" of the LLM, and this characterization helps to identify what knowledge can be externalized as explicit memory, enabling both the memory hierarchy and a lightweight backbone.

One behaviorist approach is to define the smaller parts as input-output relations between small subsequences, such that if the input text contains a subsequence belonging to some pattern, then the output text of the LLM contains a subsequence that belongs to some corresponding pattern.

- One specific input-output relation is that if the immediate context contains "China" and "capital", then output the token "Beijing".
- One abstract input-output relation is that if the immediate context is some arithmetic expression (e.g. "$123 \times 456 =$") then output the answer (e.g. "56088").
- One abstract relation that will be mentioned frequently is the "search, copy and paste" [82], such that if the context has the form "...[a][b]...[a]" then output "[b]", where [a] and [b] are arbitrary tokens.

A decomposition into these relations seems natural since autoregressive LLMs can be seen as upgraded versions of n-grams, with the fixed input/output segments generalized to flexible patterns and with the plain lookup table generalized to multi-step computations.

Nevertheless, a behaviorist approach is insufficient since an input-output relation alone cannot uniquely pin down a piece of knowledge: A LLM may answer correctly to arithmetic questions based on either the actual knowledge of arithmetic or memorization (hosting a lookup table for all expressions such as "$123 \times 456 = 56088$"). Therefore, we take a white-box approach that includes in the definition the internal computations of the LLM that convert these inputs to the related outputs.

Here are two preliminary examples of internal computations.

Example 2.1. Several works have studied the underlying mechanisms when LLMs answer to the prompt "The capital of China is" with "Beijing", as well as other factual questions [20, 26, 43, 75]. At least two mechanisms are involved, and the LLM may use their superposition [75]. One mechanism is to use general-purpose attention heads (called "mover heads") to move "capital" and "China" to the last token "is", and then use the MLP layers to map the feature of the last token to "Beijing" [75]. Often, only one or a few MLP neurons are causally relevant, and they are called "knowledge neurons" [26]. This mechanism is illustrated in Fig. 2.2 (left). Another mechanism involves attention heads h whose value-to-output matrices $W_V^h W_O^h$ function like bigrams, e.g. mapping "capital" to {"Paris", "Beijing", ...} and "China" to {"panda", "Beijing", ...} , which sum up to produce "Beijing" [20, 43, 75]. This mechanism is illustrated in Fig. 2.2 (middle).

Example 2.2. The ability of LLMs to perform "search, copy and paste", namely answering to the context "...[a][b]...[a]" with "[b]", is based on two attention heads, together called induction heads [82]. The first head copies the feature of the previous token, enabling [b] to "dress like" its previous token [a]. The second head searches for similar features, enabling the second [a] to attend to [b], which now has the appearance of [a]. Thereby, the last token [a] manages to retrieve the feature of [b] and to output [b]. This mechanism is illustrated in Fig. 2.2 (right). A similar mechanism is found for in-context learning [114].

We will address the internal mechanism for an input-output relation as a circuit, and will define a piece of knowledge as an input-output relation plus its circuit. By manipulating these circuits, one can separate many pieces of knowledge from a LLM while keeping its function intact.

Recent works on circuit discovery demonstrate that some knowledge and skills possessed by Transformer LLMs can be identified with patterns in their computation graphs

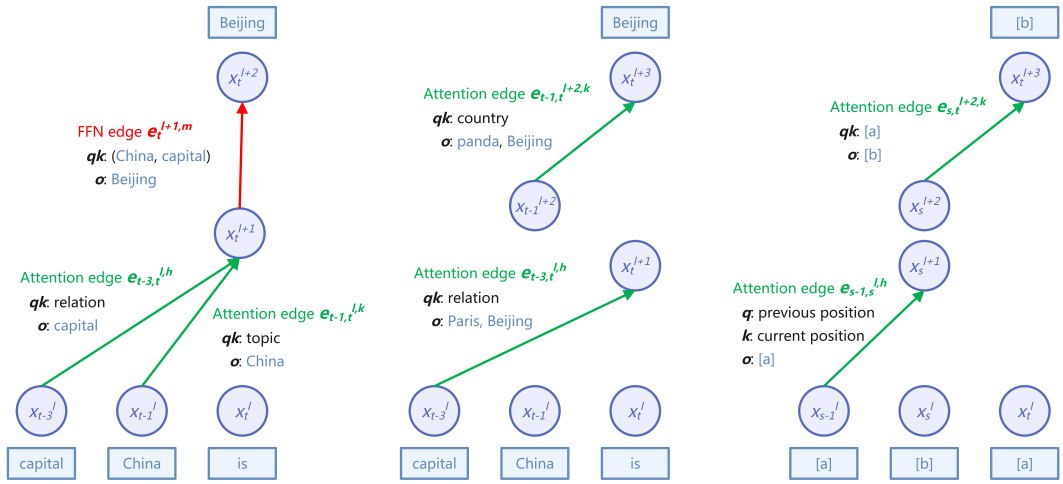

Figure 2.2: Illustration of three subgraphs. Left: A subgraph that inputs "the capital of China is" and outputs "Beijing". The knowledge neuron is marked in red and the mover heads in green. Middle: Another subgraph with similar function using task-specific heads. Right: The induction-heads subgraph that inputs "[a][b]. . . [a]" and outputs [b], where [a], [b] are arbitrary tokens. The definition of the nodes and edges is provided Definition 2.1. The exact locations of these attention heads and MLP neurons may vary.

[21, 26, 42, 43, 82, 102, 113, 114], but there has not been a universally accepted definition of circuit. Different from works on Boolean circuits [47, 77] and circuits with Transformer submodules as their nodes [21, 128], we characterize a circuit as a "spatial-temporal" phenomenon, whose causal structure is localized at the right places (MLP neurons and attention heads) and right times (tokens). Thus, we define a computation graph as a directed acyclic graph, whose nodes are the hidden features of all tokens at all MLP and attention layers, and whose edges correspond to all activations inside these layers. In particular, the computation graph hosts one copy of the Transformer architecture at each time step. To transcend this phenomenological characterization, we define a circuit as an equivalence class of similar subgraphs across multiple computation graphs.

As a remark, it is conceptually feasible to identify a circuit with the minimal subset of Transformer parameters that causes this circuit. The benefit is that such definition of knowledge seems more intrinsic to the LLM. Nevertheless, with the current definition, it is easier to perform surgery on the circuits and derive constructive proofs. Besides, it is known that Transformer submodules exhibit superposition or polysemanticity, such that one MLP neuron or attention head may serve multiple distinct functions [35, 75], making the identification of parameter subsets a challenge task.

2.2 Knowledge

We begin with the definition of the knowledge of LLMs. For now, it suffices to adopt heuristic definitions instead of fully rigorous ones. Throughout this section, by LLM we mean autoregressive Transformer LLM that has at least been pretrained. Let L be the number of Transformer blocks and H be the number of attention heads at each attention

layer, and the blocks and heads are numbered by $l = 0, \dots, L-1$ and $h = 0, \dots, H-1$. There are in total $2L$ layers (MLP layers and attention layers), and the input features to these layers are numbered by $0, \dots, 2L-1$.

Definition 2.1. *Given an LLM and a text $\mathbf{t} = (t_0, \dots t_n)$, the computation graph G on input $(t_0, \dots t_{n-1})$ and target $(t_1, \dots, t_n)$ is a directed graph with weighted edges such that*

- *Its nodes consist of the hidden vectors $\mathbf{x}_i^{2l}$ before all attention layers, the hidden vectors $\mathbf{x}_i^{2l+1}$ before all MLP layers, and the output vectors $\mathbf{x}_i^{2L}$, for all blocks $l = 0, \dots, L-1$ and positions $i = 0, \dots, n-1$.*
- *Its directed edges consist of each attention edge $e_{i,j}^{l,h}$ that goes from $\mathbf{x}_i^{2l}$ to $\mathbf{x}_j^{2l+1}$ at the h-th head of the l-th attention layer for all l, h and $i \leq j$, as well as each MLP edge $e_i^{l,m}$ that goes from $\mathbf{x}_i^{2l+1}$ to $\mathbf{x}_i^{2l+2}$ through the m-th neuron of the l-th MLP layer for all l, m, i.*
- *The weight of each attention edge $e_{i,j}^{l,h}$, which measures the influence of the attention score $a_{i,j}^{l,h}$ on the LLM output, is defined by*

$$\mathcal{L} - \mathcal{L}\big|_{a_{i,j}^{l,h}=0} \quad or \quad \frac{\partial \mathcal{L}}{\partial a_{i,j}^{l,h}},$$

  where $\mathcal{L}$ is the log-likelihood of the target $(t_1, \dots, t_n)$, with $\mathcal{L}|_{a=0}$ obtained by setting $a = 0$ (i.e. causal intervention). Similarly, the weight of each MLP edge $e_i^{l,m}$, which measures the influence of the neuron activation $a_i^{l,m}$ on the LLM output, is defined likewise.
- *Given any subgraph $S \subseteq G$, define the associated input of S as a subsequence $\mathbf{t}_{\text{in}}(S) \subseteq (t_0, \dots, t_{n-1})$ such that a token t_i belongs to $\mathbf{t}_{\text{in}}(S)$ if and only if $\|\nabla_{\mathbf{x}_i^0} a\|$ is large for some attention edge (or MLP edge) in S with attention score (or activation) a.*
- *Similarly, define the associated output of the subgraph S as a subsequence $\mathbf{t}_{\text{out}}(S) \subseteq (t_1, \dots, t_n)$ such that a token t_i belongs to $\mathbf{t}_{\text{out}}(S)$ if and only if*

$$\mathcal{L}_i - \mathcal{L}_i\big|_{a=0} \quad or \quad \frac{\partial \mathcal{L}_i}{\partial a}$$

  is large for some attention edge (or MLP edge) in S with attention score (or activation) a. Here $\mathcal{L}_i$ is the log-likelihood of t_i with respect to the LLM output.

Definition 2.2. *Given two computation graphs G_1, G_2 of an LLM and their subgraphs S_1, S_2, a mapping f from the nodes of S_1 to the nodes of S_2 (not necessarily injective) is a homomorphism if*

- *every node at depth $l \in \{0, \dots, 2L\}$ is mapped to depth l,*
- *if two nodes are on the same position i, then they are mapped onto the same position,*
- *if two nodes share an edge on attention head h or MLP neuron m, then their images also share an edge on head h or neuron m.*

If such a homomorphism exists, then we say that S_1 is homomorphic to S_2.

An illustration of computation graph and homomorphism is provided in Fig 2.3. It may be more convenient to define the mapping to be between the input tokens of two sentences, but we adopt the current formulation as it is applicable to more general settings without an obvious correspondence between the tokens and the hidden features at each layer.

Definition 2.3. *Given an LLM and a distribution of texts, a circuit is an equivalence class $\mathcal{K}$ of subgraphs from computation graphs on random texts such that*

- *The computation graph on a random text contains some subgraph $S \in \mathcal{K}$ with positive probability.*
- *All subgraphs $S \in \mathcal{K}$ are homomorphic to each other.*
- *All edges of all $S \in \mathcal{K}$ have non-negligible weights.*
- *The pairs $(\mathbf{t}_{\text{in}}(S), \mathbf{t}_{\text{out}}(S))$ share some interpretable meaning across all $S \in \mathcal{K}$.*

Definition 2.4. *Given an LLM and a distribution of texts, we call each circuit a knowledge. Furthermore, a circuit $\mathcal{K}$ is called a*

- *specific knowledge, if the associated inputs $t_{\text{in}}(S)$ for all subgraphs $S \in \mathcal{K}$ share some interpretable meaning, and the associated outputs $t_{\text{out}}(S)$ for all $S \in \mathcal{K}$ are the same or differ by at most a small fraction of tokens,*
- *abstract knowledge, else.*

From now on, we use knowledge as a countable noun since the circuits are countable. Note that the criterion in Definition 2.4 is stronger than the last criterion in Definition 2.3, e.g. consider the circuit that always copy-and-pastes the previous token. We will see that the rigidity of specific knowledge makes them easier to externalize.

Here are some well-known examples of knowledge.

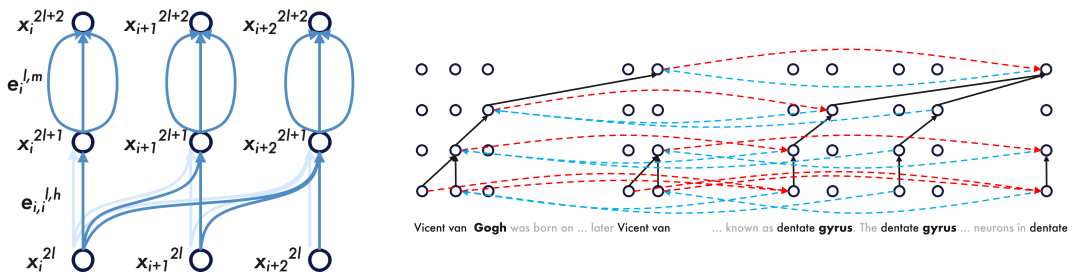

Figure 2.3: Left: Illustration of the computation graph over one Transformer block, showing only three tokens, one attention head and three MLP neurons. The edge weights are not shown. Right: The subgraphs S_1, S_2, namely the induced subgraphs of the attention edges (black arrows), belong to the circuit of the induction head. The red arrows denote a homomorphism from S_1 to S_2, and the blue arrows denote a homomorphism from S_2 to S_1.

Example 2.3. Recall the knowledge neuron from Example 2.1 that helps to answer "The capital of China is Beijing". Such neurons can be activated by a variety of contexts that involve the subject-relation pair ("China", "capital") [26]. Its circuit can be simply defined as the equivalence class of subgraphs induced by edges $e_{i,\cdot}^{l,m}$, where (l, m) is the fixed location of the knowledge neuron and i is the variable position of the last token of the context. The associated inputs are "China" and "capital", and the associated outputs are always "Beijing". By definition, this circuit is a specific knowledge, since its associated output is fixed and its associated inputs share a clear pattern (fixed tokens with variable positions).

Similarly, by straightforward construction, one can show that each n-gram can be expressed as a specific knowledge.

Example 2.4. Recall the induction heads [82] from Example 2.2 that complete "[a][b] . . . [a]" with "[b]". Let $(l, h), (l + 1, h')$ be the locations of these two heads, and denote the variable positions of the two token [a]'s by i, j. Its circuit is the equivalence class of subgraphs induced by the two edges $e_{i,i+1}^{l,h}, e_{i+1,j}^{l+1,h'}$. Although the associated input-output pairs "[a][b]. . . [a][b]" have a clear pattern, the associated outputs "[b]" alone can be arbitrary, so the induction head is an abstract knowledge.

More sophisticated abstract knowledge have been identified for in-context learning [114] and indirect object identification [113].

Definition 2.5. *Given a LLM and a knowledge $\mathcal{K}$, a text $\mathbf{t} = (t_0, \ldots, t_n)$ is called a realization of $\mathcal{K}$, if the computation graph on $\mathbf{t}$ has a subgraph that belongs to $\mathcal{K}$.*

For instance, any text of the form [a][b]. . . [a][b] can be a realization of the abstract knowledge of induction head.

Our definition of knowledge is extrinsic, depending on a specific LLM, instead of intrinsic, depending only on texts. From this perspective, problem (1.1) can be interpreted as relocating the knowledge from an all-encompassing LLM to more efficient models equipped with memory hierarchy. For concreteness, one can fix this reference LLM to be the latest version of ChatGPT or Claude [2, 6], or some infinitely large model from a properly defined limit that has learned from infinite data.

Assumption 2.1 (Completeness)**.** Fix a reference LLM and a distribution of texts, let G be the computation graph of a random text. Assume that there exists a set $\mathfrak{K}$ of knowledge such that, with probability 1 over the random text, the subgraph of G induced by edges with non-negligible weights can be expressed as a union of subgraphs $\{S_i \in \mathcal{K}_i\}$ from $\{\mathcal{K}_i\} \subseteq \mathfrak{K}$.

Essentially, Assumption 2.1 posits that all computations in the LLM can be fully decomposed into circuits, so that the LLM is nothing more than a collection of specific and abstract knowledge. This viewpoint underscores that the efficiency of LLMs is ultimately about the effective organization of these knowledge, an objective partially addressed by problem (1.1).

2.3 Memory

Now the question is what knowledge can be separated from the model parameters and moved to the lower levels of the memory hierarchy.

Definition 2.6. *A knowledge $\mathcal{K}$ of the reference LLM is separable if there exists another LLM M such that*

- *M does not possess this knowledge such that for any realization $\mathbf{t}$ of $\mathcal{K}$, the model M cannot generate each token of the associated output $\mathbf{t}_{\text{out}}$ with high probability, e.g. $\mathbb{P}_M(t_i|t_0 \dots t_{i-1}) \leq 1/2$ for some $t_i \in \mathbf{t}_{\text{out}}$.*
- *There exists a text $\mathbf{t}_*$ such that for any realization $\mathbf{t}$ of $\mathcal{K}$, the model M using $\mathbf{t}_*$ as prefix can generate each token of the associated output $\mathbf{t}_{\text{out}}$ with high probability, e.g. $\mathbb{P}_M(t_i|\mathbf{t}_* t_0 \dots t_{i-1}) \geq 0.9$ for every $t_i \in \mathbf{t}_{\text{out}}$.*

If among the realizations of $\mathcal{K}$, the same associated input $\mathbf{t}_{\text{in}}$ can correspond to multiple associated outputs $\mathbf{t}_{\text{out}}$, then the above probabilities are summed over all branches if position i is a branching point.

Definition 2.7. *A separable knowledge $\mathcal{K}$ of the reference LLM is imitable if any realization $\mathbf{t}'$ of $\mathcal{K}$ can be used as the prefix $\mathbf{t}_*$ in Definition 2.6, e.g. for any realizations $\mathbf{t}, \mathbf{t}'$ of $\mathcal{K}$, we have $\mathbb{P}_M(t_i|\mathbf{t}' t_0 \dots t_{i-1}) \geq 0.9$ for every $t_i \in \mathbf{t}_{\text{out}}$.*

Let us note that imitability basically means that LLMs can achieve the same effect as possessing this knowledge by retrieving example texts that demonstrate this knowledge. Few-shot prompting can be seen as a special case of providing realizations.

Separability is a more general property than imitability. For instance, one can set the prefix $\mathbf{t}_*$ to be an abstract description of $\mathcal{K}$ instead of its realization, and this is reminiscent of instruction prompting. Nevertheless, it is not obvious whether the set of separable knowledge is strictly larger than the set of imitable knowledge.

Claim 2.1. Every specific knowledge $\mathcal{K}$ is imitable and thus is separable.

Proof (informal). Without loss of generality, we can assume that for any realization $\mathbf{t}$ of $\mathcal{K}$, all tokens of the associated input $\mathbf{t}_{\text{in}}$ precede all tokens of the associated output $\mathbf{t}_{\text{out}}$. Otherwise, we can split $\mathbf{t}_{\text{in}}$ into two halves $\mathbf{t}_1, \mathbf{t}_2$ that precedes/does not precede $\mathbf{t}_{\text{out}}$, and split the corresponding subgraph $S \in \mathcal{K}$ into two halves S_1, S_2 that have high weights with respect to $\mathbf{t}_1, \mathbf{t}_2$. Using monotonicity arguments once Definition 2.3 is fully formalized, one can try to show that this splitting is invariant across $S \in \mathcal{K}$ and therefore the sets of S_1, S_2 are two specific knowledge.

Consider sequences of the form [a][b]...[a'][b'], where [a], [a'] (or [b], [b']) could be the associated inputs (or outputs) of any subgraphs $S, S' \in \mathcal{K}$. By Definition 2.4, [a] and [a'] always share some interpretable meaning, while [b] and [b'] are approximately the same sequence. One can construct an abstract knowledge that completes [a][b]...[a'] with [b']: The first part of this circuit detects the common feature of the [a]'s (possibly overlapping with the subgraphs of $\mathcal{K}$), the second part is an induction head (analogous to Example 2.4, it provides [b] with the common feature of the [a]'s and lets [a'] to attend to [b]), and the

third part generates [b′] based on [b] with possible slight modifications. This circuit is an abstract knowledge since it can be applied to other specific knowledges as long as their associated inputs share the same meaning with the [a]'s, no matter how their associated outputs could vary.

Meanwhile, construct the model M by letting the reference model forget $\mathcal{K}$ (e.g. by finetuning on a modified data distribution such that the associated input of $\mathcal{K}$ is never followed by the associated output, while the rest of the distribution remains the same). Combining this circuit with M completes the proof. □

Claim 2.1 indicates that a lot of knowledges can be externalized from the model parameters. The converse of Claim 2.1 may not hold, since it is imaginable that some abstract knowledges can also be substituted with their realizations.

Remark 2.1. There are three details in the proof of Claim 2.1 that will be useful later:

1. The circuit we construct has only one attention head that attends to the reference text $\mathbf{t}'$ from the present text $\mathbf{t}$, while all other computations are confined within either $\mathbf{t}$ or $\mathbf{t}'$.
2. Moreover, in this attention head, the circuit only needs the edges from [b] to [a′]. Thus, in general this head only needs to attend to very few tokens in the reference.
3. It suffices for the reference $\mathbf{t}'$ to attend only to itself.

These properties will guide our architecture design.

To finish the set-up of problem (1.1), we define the memory formats. The definition should subsume the aforementioned formats of model parameters, explicit memories and plain texts for RAG, and also allow for new memory formats of future LLMs.

Definition 2.8. *Let $\mathfrak{K}$ be the complete set of knowledges from Assumption* 2.1 *and consider the subset of separable knowledges. Let $\mathfrak{T}$ be a set that contains one or several realizations $\mathbf{t}$ for each separable knowledge. Let $f_1, \dots, f_m$ be any functions over $\mathfrak{T}$. Abstractly speaking, a memory-augmented LLM M is some mapping from prefixes to token distributions with additional inputs*

$$M : \big((t_0, \dots, t_{i-1}), \{\mathcal{K}_1, \dots, \mathcal{K}_N\}, X_1, \dots, X_m\big) \;\mapsto\; \mathbb{P}(\cdot|t_0 \dots t_{i-1}), \tag{2.1}$$

where the set $\{\mathcal{K}_1, \dots, \mathcal{K}_N\}$ consists of non-separable knowledges of M that are invoked at this step, and the sets X_j consist of encoded texts

$$X_j = \{f_j(\mathbf{t}_{j,k})\} \tag{2.2}$$

for some $\mathbf{t}_{j,k} \in \mathfrak{T}$.

Each $j = 1, \dots, m$ represents a memory format and f_j is called the `write` *function of this format. If some realization of a separable knowledge $\mathcal{K}$ participates in the mapping M, then we say that $\mathcal{K}$ is* `written` *in format j and* `read` *by M.*

Analogous to Assumption 2.1, we are decomposing each step of LLM inference into the invoked circuits, but the decomposition here also involves reference texts that are written in various memory formats.

Table 1.1 demonstrates that the write functions could be diverse, and the list is probably far from conclusive. Nevertheless, some heuristics still apply. The write function f_j and the read process in M for each format j should be non-trivial such that, for any separable knowledge $\mathcal{K}$ not contained in M and any realization $\mathbf{t}$ of $\mathcal{K}$, if $\mathcal{K}$ enters in M through format j, then M should be able to generate each token of the associated output of $\mathcal{K}$ in $\mathbf{t}$ with higher probability as in Definition 2.6. Thus, informally speaking, the total cost of writing and reading $\mathcal{K}$ must be bounded from 0, since some minimum computation is necessary for reducing the uncertainty in generating the correct tokens. It follows that the write cost and read cost are complementary, i.e. cheaper writing must be accompanied by more expensive reading.

We define this inverse relationship between the write cost and read cost as the memory hierarchy. This relationship is in accordance with our experience regarding the three examples of human memories in Table 1.1, e.g. we can utter the common expressions almost immediately while it may take a few seconds to recall a book we read, but the former skill is acquired through years of language speaking. For the LLM memories in Table 1.1, the inverse relationship is illustrated Fig. 1.1 and established by the calculations in Appendix A.

The imbalanced use of knowledges leads to a heterogeneous distribution of knowledges across the memory hierarchy. To minimize the total cost (1.1), the separable knowledges that are used more often should be assigned to memory formats with high write cost and low read cost, whereas the rarely used knowledges should be assigned to formats with low write cost and high read cost. Also, adding a new memory format $m+1$ is always beneficial as it expands the search space and decreases the minimum cost whenever the usage count of some knowledge $\mathcal{K}$ lies in the interval

$$[n^-_{m+1}, n^+_{m+1}] = \left\{ n \in [0,\infty) \;\middle|\; \operatorname{argmin}_j \operatorname{cost}_{\text{write}}(\mathcal{K}, j) + n \cdot \operatorname{cost}_{\text{read}}(\mathcal{K}, j) = m+1 \right\}.$$

Examples of these intervals are displayed in Fig. 1.1. For concreteness, Fig. 2.4 depicts a reasonable distribution of the specific knowledges for humans, and we expect a similar distribution to hold for LLMs equipped with explicit memory.

3 Design

This section describes the architecture and training scheme of Memory^3.

Regarding architecture, the goal is to design an explicit memory mechanism for Transformer LLMs with moderately low write cost and read cost. In addition, we want to limit the modification to the Transformer architecture to be as little as possible, adding no new trainable parameters, so that most of the existing Transformer LLMs can be converted to Memory^3 models with little finetuning. Thus, we arrive at a simple design:

- Write cost: Before inference, the LLM writes each reference to an explicit memory, saved on drives. The memory is selected from the key-value vectors of the self-attention layers, so the write process involves no training. Each reference is processed independently, avoiding the cost of long-context attention.

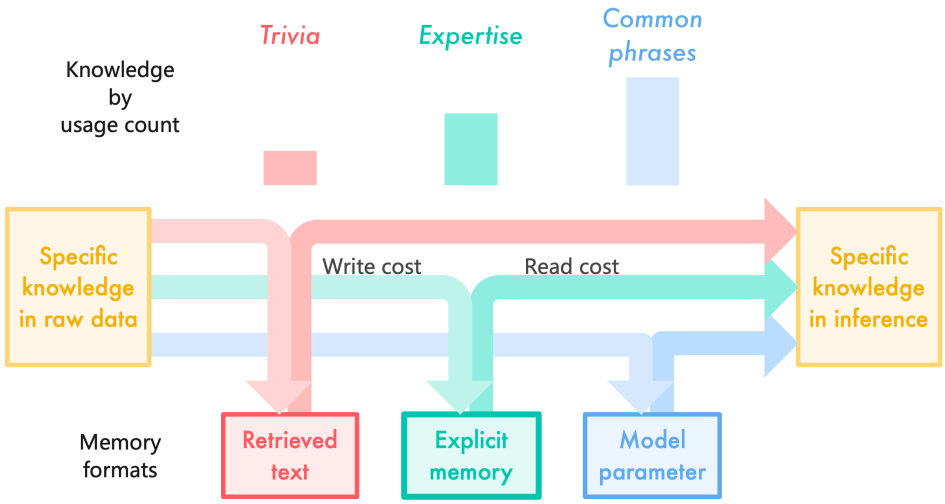

Figure 2.4: Different memory formats with different balances of write cost and read cost. The specific knowledges with high to low usage counts are exemplified by common expressions, expertise and trivia, and are assigned to implicit memory, explicit memory and external information.

- Read cost: During inference, explicit memories are retrieved from drives and read by self-attention alongside the usual context key-values. Each memory consists of very few key-values from a small amount of attention heads, thus greatly reducing the extra compute, GPU storage, drive storage and loading time. It allows the LLM to retrieve many references frequently with limited influence on decoding speed.

Regarding training, the goal is to reduce the cost of pretraining with a more efficient distribution of knowledge. Based on the discussion in Section 2.3, we want to encourage the LLM to learn only abstract knowledges, with the specific knowledges mostly externalized to the explicit memory bank. Ideally, the pretraining cost should be reduced to be proportional to the small amount of knowledge stored in the model parameters, thereby taking a step closer to the learning efficiency of humans.

3.1 Inference process

From now on, we refer to the realizations of separable knowledges (Definitions 2.5 and 2.6) as references. Our knowledge base (or reference dataset) consists of 1.1×10^8 text chunks with length bounded by 128 tokens. Its composition is described in Section 4.4.

Each reference can be converted to an explicit memory, which is a tensor with shape

$$(\text{memory layers}, 2, \text{key-value heads}, \text{sparse tokens}, \text{head dimension}) = (22, 2, 8, 8, 80).$$

The 2 stands for the key and value, while the other numbers are introduced later.

Before inference, the Memory^3 model converts all references to explicit memories and save them on drives or non-volatile storage devices. Then, at inference time, whenever (the id of) a reference is retrieved, its explicit memory is loaded from drives and sent to GPU to be integrated into the computation of Memory^3. By Remark 2.1, a reference during

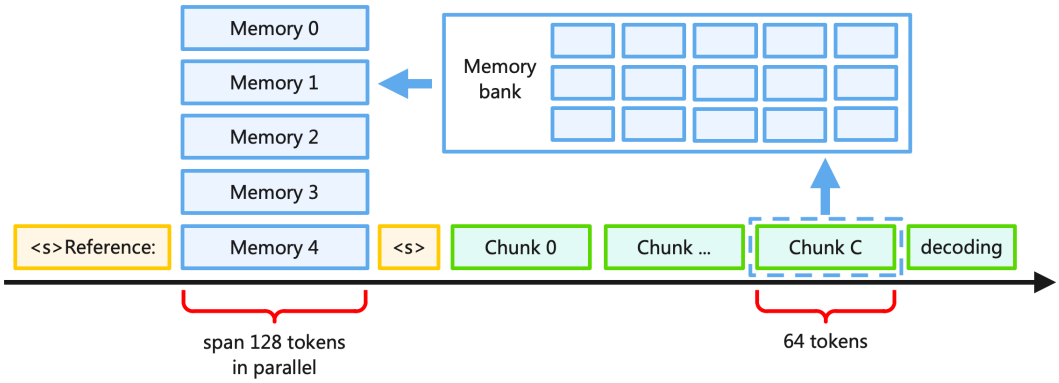

Figure 3.1: The decoding process of Memory^3 with memory recall. Each chunk is a fixed-length interval of tokens, which may belong to either the prompt or generated text.

encoding does not need to attend to any other texts (e.g. other references or query texts), so it is fine to encode each reference independently prior to inference. Such isolation also helps to reduce the compute of attention.

One can also employ a "cold start" approach to bypass preparation time: each reference is converted to explicit memory upon its initial retrieval, rather than prior to inference. Subsequent retrievals will then access this stored memory. The aforementioned inference with precomputed explicit memories will be called "warm start".

During inference, as illustrated in Fig. 3.1, whenever the LLM generates 64 tokens, it discards the current memories, uses these 64 tokens as query text to retrieve 5 new memories, and continues decoding with these memories. Similarly, when processing the prompt, the LLM retrieves 5 memories for each chunk of 64 tokens. Each chunk attends to its own memories, and the memories could be different across chunks. We leave it to future work to optimize these hyperparameters.

The retrieval is performed with plain vector search with cosine similarity. The references as well as the query chunks are embedded by BGE-M3, a multilingual BERT model [15]. The query and key vectors for retrieval are both obtained from the output feature of the $\langle$cls$\rangle$ token. The vector index is built with FAISS [32].

To further save time, we maintain a fixed-size cache in RAM to store the most recently used explicit memories. It is been observed that adjacent chunks often retrieve some of the same references. So the cache reduces the cost of loading explicit memories from drives.

Remark 3.1. It would be ideal to perform retrieval using the hidden features from the LLM itself, since conceptually the LLM should know its needs better than any external module, and such internalized retrieval appears more anthropomorphic. Moreover, retrieving with the hidden features from different layers, different heads and different keywords can help to obtain more diverse results. One simple implementation is to use the sparsified attention queries of the query text to directly search for the explicit memories. Since the explicit memories are the attention key-values, such retrieval can work without the need to fine-tune the LLM. Specifically, this multi-vector retrieval can follow the routine of [58] with the additional constraint that a query from attention head h can only search for keys from h,

while the sparse attention queries can be obtain using the same selection mechanism for explicit memories described later.

Remark 3.2. One shortcoming of RAG is that the references are usually text chunks instead of whole documents, and thus during inference the references are encoded without their contexts, making them less comprehensible. This shortcoming can be easily overcome for explicit memories. One solution is to encode each document as one sequence, then chunk the attention key-values into 128-token chunks and sparsify them into explicit memories. This procedure allows the key-values to attend to all their contexts.

3.2 Writing and reading memory

Each explicit memory is a subset of the attention key-values from a subset of attention heads when encoding a reference. Thus, during inference, the LLM can directly read the retrieved explicit memories through its self-attention layers by concatenating them with the usual context key-values (Fig. 3.1). Specially, for each attention head h at layer l, if it is chosen as a memory head, then its output $Y^{l,h}$ changes from the usual

$$Y_i^{l,h} = \text{softmax}\left(\frac{X_i^l W_Q^{l,h} \left(X_{[:i]}^l W_K^{l,h}\right)^\top}{\sqrt{d_h}}\right) X_{[:i]}^l W_V^{l,h} W_O^{l,h},$$

where $X_{[:i]}$ denotes all tokens before or at position i and d_h denotes the head dimension, to

$$\begin{aligned} Y_i^{l,h} = \text{softmax}&\left(\frac{X_i^l W_Q^{l,h} \cdot \text{concat}\left(K_0^{l,h}, \ldots, K_4^{l,h}, X_{[:i]}^l W_K^{l,h}\right)^\top}{\sqrt{d_h}}\right) \\ &\times \text{concat}\left(V_0^{l,h}, \ldots, V_4^{l,h}, X_{[:i]}^l W_V^{l,h}\right) W_O^{l,h}, \end{aligned} \tag{3.1}$$

where each (K_j, V_j) denotes the keys and values of an explicit memory.

Regarding the beginning-of-sentence token `BOS`, it is set to be the usual $\langle \text{s} \rangle$ symbol when it is placed at the beginning of the context. However, we set the `BOS` at the beginning of each reference to "$\langle \text{s} \rangle$Reference:" to help the LLM distinguish between encoding the context and encoding references. As illustrated in Fig. 3.1, this modified `BOS` is the actual beginning for both training and inference, while the context `BOS` token serves as a separator between the references and context. Unlike the explicit memories which only appear at a subset of attention heads, this modified `BOS` is placed at every head at every layer even if there are no references. The motivation is that since the context `BOS` can attend to the references, its feature is no longer constant, so the LLM needs the modified `BOS` to serve as the new constant for all attention heads.

Furthermore, we adopt parallel position encoding for all explicit memories, namely the positions of all their keys lie in the same interval of length 128, as depicted in Fig. 3.1. We use the rotary position encoding (RoPE) [103]. The token sparsification is applied after RoPE processes the attention keys, so the selected tokens retain their relative positions in the references. Besides flexibility, one motivation for parallel position is to avoid the "lost

in the middle" phenomenon [68], such that if the references are positioned serially, then the ones in the middle are likely to be ignored. Similarly, token sparsification also helps to alleviate this issue by making the attention more focused on the important tokens. We note that designs analogous to the parallel position have been used to improve in-context learning [93] and long-context modeling [13].

3.3 Memory sparsification and storage

One of the greatest challenges for explicit memories is that the attention key-values occupy too much space. They not only demand more disk space, which could be costly, but also occupy GPU memory during inference, which could harm the batch size and thus the throughput of LLM generation. An intense compression is needed to save space. The full attention key tensor (or value tensor) for each reference has shape (layers, key-value heads, tokens, head dimension), so we compress all four dimensions.

Regarding layers, we only set the first half of the attention layers to be memory layers, i.e. layers that produce and attend to explicit memories (3.1), while the second half remain as the usual attention layers. Note that Remark 2.1 suggests that it is usually the attention heads in the middle of the LLM that attend to the references. So it seems that appointing the middle attention layers (e.g. the ones within the 25% to 75% depth range) to be memory layers is a more sensible choice. This heuristic is supported by the observations in [36,121] that the attention to the distant context usually takes place in the middle layers.

Regarding heads, we set all key-value heads at each memory layer to be memory heads. We reduce their amount by grouped query attention (GQA) [3], letting each key-value head be shared by multiple query heads, and obtain 20% sparsity (8 versus 40 heads). It is worth mentioning that, besides GQA and memory layers, another approach is to select a small subset of heads that are most helpful for reading memories, and this selection does not have to be uniform across layer. We describe several methods for selecting memory heads in Remark 3.3.

Regarding tokens, we select 8 tokens out of 128 for each key-value head. We choose a high level of sparsity, since Remark 2.1 indicates that the attention from the context to the references are expected to be concentrated on very few tokens. Note that the selected tokens are in general different among heads, so in principle their union could cover a lot of tokens. For each head h at layer l, the selection uses top-8 over the attention weight

$$w_j^{l,h} = \sum_{i=0}^{127} \tilde{a}_{i,j}^{l,h}, \quad \tilde{a}_{i,j}^{l,h} = \mathrm{softmax}_j\left(\frac{X_i^l W_Q^{l,h} \left(X_j^l W_K^{l,h}\right)^\top}{\sqrt{d_h}}\right),$$

which measures the importance of a token by the attention received from all tokens. The `BOS` tokens and paddings do not participate in the computation of the weights. These attention weights $\tilde{a}$ are different from the usual ones, such that there is no causal mask or position encoding involved. The consideration is that since the explicit memories are prepared before any inference, the selection can only depend on the reference itself instead of any context texts. The removal of causal mask and position encoding ensures that tokens at any position has an equal chance to receive attention from others. To speed

up computation, we adopt the following approximate weights in our implementation, although in retrospect this speedup is not necessary:

$$w_j^{l,h} = \sum_{i=0}^{127} \exp\left(\frac{X_i^l W_Q^{l,h} \left(X_j^l W_K^{l,h}\right)^\top}{\sqrt{d_h}}\right).$$

Similar designs that sparsify tokens based on attention weights have been adopted in long-context modeling to save space [70, 131].

Regarding head dimension, we optionally use a vector quantizer to compress each of the key and value vectors using residual quantizations [16] built with FAISS [32]. The compression rate is $80/7 \approx 11.4$. During inference, the retrieved memories are first loaded from drives, and then decompressed by the vector quantizer before being sent to GPU. The evaluations in Section 7.1 indicate that this compression has negligible influence on the performance of Memory^3. More details can be found in Appendix B.

Hence, the total sparsity is 160 or 1830 (without or with vector compression). Originally, the explicit memory bank would have an enormous size of 7.17 PB or equivalently 7340 TB (given the model shape described in Section 3.4 and saved in bfloat16). Our compression brings it down to 45.9 TB or 4.02 TB (without or with vector compression), both acceptable for the drive storage of a GPU cluster.

To deploy the Memory^3 model on end-side devices such as smart phones and laptops, one can place the explicit memory bank and the vector index on a cloud server, while the devices only need to store the model parameters and the decoder of the vector quantizer. During inference, to perform retrieval, the model on the end-side device sends the query vector to the cloud server, which then searches the index and returns the compressed memories. The speed test of this deployment is recorded in Section 7.5.

Remark 3.3. If one wants to finetune a pretrained LLM into a Memory^3 model, there are several ways to select a small but effective subset of attention heads (among all heads at all layers) for memory heads (3.1). Methods such as [36, 121] are proposed to identify the heads that contribute the most to long-context modeling by retrieving useful information from distant tokens, and usually these special heads account for only less than 10% of the total heads. Here we also propose a simple method for selecting memory heads: Given the validation subsets of a representative collection of evaluation tasks, one can measure the average performance s_h for a modified version of the LLM for each attention head h. The modification masks the distant tokens for head h so it can only see the preceding 100 tokens and the `BOS` token. Then, it is reasonable to expect that s_h would be markedly low for a small subset of heads h, indicating that they are specialized for long-range attention.

Remark 3.4. Actually, Remark 2.1 suggests that each reference only needs to be attended to by just one attention head, although in general this special head may be different among the references. Thus, it seems a promising approach to apply adaptive sparsity not only to token selection, but also to the memory heads, namely each reference is routed to one or two heads (analogously to MoE), and its explicit memory is produced and read by these heads. Such design if feasible can further boost the sparsity of explicit memory and save much more space.

3.4 Model shape

As discussed in Section 2.3, the specific knowledges can be externalized to explicit memories, and thus to minimize the total cost (1.1), the model parameters (or implicit memory) only need to store abstract knowledges and the subset of specific knowledges that are frequently used. The shape of our model, i.e. (the number of Transformer blocks L, heads H, head dimension d_h, width of the MLP layers W), is chosen to accommodate this desired knowledge distribution. Informally speaking, given a fixed parameter size P, the shape maximizes the following objective:

$$\max_{L,H,d_h,W} \left\{ \frac{\text{capacity for abstract knowledge}}{\text{capacity for specific knowledge}} \;\middle|\; \text{size}(L,H,d_h,W) \approx P \right\}. \tag{3.2}$$

Here we set P to be 2.4 billion.

Some recent works suggest that the capacities for learning specific knowledges and abstract knowledges are subject to different constraints. On one hand, [26] observes that the amount of bits of trivia information (such as a person's name, date of birth and job title) that a LLM can store depends only on its parameter size. Regardless of L and H, the max capacity is always around 2 bits per parameter.

On the other hand, [118] trains Transformers to learn simple algorithms such as reversing a list and counting the occurrence of each letter. It is observed that for several such tasks, there exists a minimum L_0 and H_0 such that a Transformer with $L \geq L_0$ and $H \geq H_0$ can learn the task with perfect accuracy, whereas the accuracy drops significantly for Transformers with either $L = L_0 - 1$ or $H = H_0 - 1$ (given that either L_0 or $H_0 \geq 2$). This sharp transition supports the view that the layers and heads of Transformer LLMs can be compared to algorithmic steps, and tasks with a certain level of complexity require at least a certain amount of steps. It is worth mentioning that the emergent phenomenon [101, 117] of LLMs can also be explained by this view and thus adds support to it, although it may not be the only explanation.

By Definition 2.4, the abstract knowledges are expected to be circuits with greater complexity than specific knowledges, since their associated inputs and outputs exhibit greater variability and thus express more complex patterns. It follows that, in the context of the aforementioned works, the separation of specific and abstract knowledges should be positively correlated with the distinction between trivia information and algorithmic procedures. Hence, it is reasonable to adopt the approximation that the capacity of an LLM for specific knowledges only depends on its parameter size, whereas the capacity for abstract knowledges depends only on L and H.

The informal problem (3.2) reduces to the maximization of L and H given a fixed parameter size. However, we are left with two ambiguities: First, this formulation does not specify the ratio between L and H, and second the head dimension d_h and MLP width W cannot be too small as the training may become unstable. Regarding the second point, our experiments indicate that pretraining becomes more unstable with increased spikes if $d_h \leq 64$, so we set $d_h = 80$ (though it needs to be pointed out that the loss spikes may not be solely attributed to the choice of d_h, and high-quality data for instance may stabilize training and allow us to choose a smaller d_h). Also, the MLP width W is set to be

equal to the hidden dimension $d = Hd_h$. Regarding the first point, controlled experiments (Fig. 3.2) indicate that the loss decreases slightly more rapidly with $L : H \approx 1$ than with other ratios, so we adopt this ratio.

In addition, as discussed in Section 3.3, our model uses grouped query attention (GQA), so the number of key-value heads H_{kv} is set to be 8, which is the usual choice for GQA. The MLP layers are gated two-layer networks without bias, which are the default choice in recent years [5, 9, 19, 108].

Finally, the model shape is set to be $L = 44, H = 40, H_{kv} = 8, d_h = 80, W = 3200$, with the total non-embedding parameter size being 2.4B.

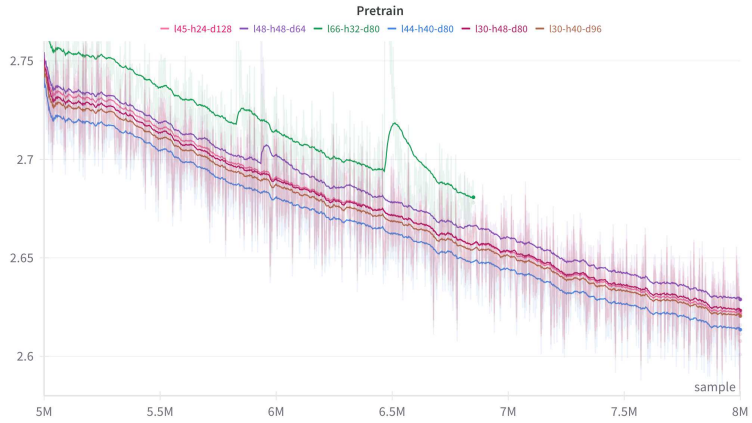

Figure 3.2: Comparison of the training losses of models with different shapes, whose parameter sizes range in 2.1 $\sim$ 2.4B. The legend l44h40d80 denotes $L = 44, H = 40, d_h = 80$, and the x-axis denotes the amount of training samples. Nevertheless, this comparison is not definite, since this is only the implicit memory stage of our training scheme (Section 3.6) and the ranking may change in the explicit memory stage that follows.

3.5 Training designs

Similar to our architecture design, the design of our training scheme focuses on learning abstract knowledges. The goal is to reduce the training compute, as the LLM no longer needs to memorize many of the specific knowledges. This shift in learning objective implies that all the default settings for pretraining LLMs may need to be redesigned, as they were optimized for the classical scenario when the LLMs learn both abstract and specific knowledges.

1. Data: Ideally, the pretraining data should have a high concentration of abstract knowledges and minimum amount of specific knowledges. It is known that LLM pretraining is very sensitive to the presence of specific knowledges. For instance, [52] observes that a small model can master arithmetic (e.g. addition of large numbers) if trained on clean data. However, if the training data is mixed with trivial information (e.g. random numbers), then the test accuracy stays at zero unless the model size is increased by a factor of 1500. It suggests that training on specific knowledges significantly inhibits the learning of abstract knowledges, and may explain why emer-

gent abilities [117] are absent from small models. Notably, the Phi-3 model [1] is pretrained with a data composition that closely matches our desired composition. Although the technical details are not revealed, it is stated that they filter data based on two criteria: the data should encourage reasoning, and should not contain information that is too specific.

2. Initialization: [130] observes that initializing Transformer parameters with a smaller standard deviation (d^c with $c < -1/2$ instead of the usual $\Theta(d^{-1/2})$ [44, 49]) can encourage the model to learn compositional inference instead of memorization. Specially, an arithmetic dataset is designed with a train set and an out-of-distribution test set, which admits two possible answers. One answer relies on memorizing more rules during training, while the other requires an understanding of the compositional structure underlying these rules. The proposed mechanism is that training with smaller initialization belongs to the condensed regime that encourages sparse solutions, contrary to training with large initialization that belongs to the kernel regime or critical regime [17,74].

3. Weight decay: [85,87] observe that using a larger weight decay coefficient (i.e. greater than the usual range of $0.001 \sim 0.1$) can guide LLMs to favor generalization over memorization, and accelerate the learning of generalizable solutions. They consider settings that exhibit grokking [87] such that training would transit from perfect train accuracy and zero test accuracy to perfect test accuracy, and generalization ability is measured by how quickly this transition occurs. Moreover, theoretically speaking, it is expected that training generative models needs stronger regularization than training regression models, in order to prevent the generated distributions from collapsing onto the training data and become trivial [127].

In summary, it is recommendable to pretrain the Memory^3 model with a data composition that emphasizes abstract knowledges and minimizes specific information, a smaller initialization for parameters, and a larger weight decay coefficient.

Since this work is only a preliminary version of Memory^3, we decide to stick with the conventional setting for training and have not experimented with any of these ideas. We look forward to incorporating these designs in future versions of the Memory^3 model.

3.6 Two-stage pretrain

The Memory^3 model learns to write and read explicit memories during pretraining. Each training sample is accompanied by retrieved references, the model encodes these references into explicit memories and integrates them into the self-attention of the training sequence. The format is identical to that of inference (Fig. 3.1), except that the memories are made in real time. Then, the loss is computed over the tokens of the training sequence, ignoring the references.

Our pretraining consists of two stages, the implicit memory stage and explicit memory stage. Only the second stage involves explicit memories, while the implicit memory stage follows the same format as ordinary pretraining. Our motivation is depicted in Fig. 3.3.

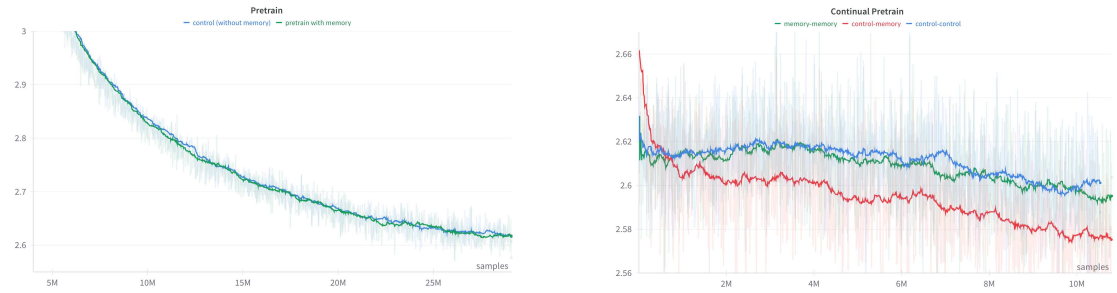

Figure 3.3: Left: Comparison of pretraining with and without explicit memory. The blue/green curves are trained without/with explicit memories. Right: Comparison of continual pretraining. The blue/green curves are initialized with their checkpoints in the left plot and continually pretrained without/with explicit memories, while the red curve is initialized with the checkpoint of the blue curve and continually pretrained with explicit memory. These plots indicate that pretraining a Memory^3 model requires an implicit-memory-only stage. These experiments use a smaller model with 0.92 B non-embedding parameters $(L = 40, H = 32, d_h = 64)$. The left plot uses 60 B data while the right plot uses 22 B. The same learning rate schedule is adopted in each plot.

We observe that pretraining with explicit memories from the beginning would render the memories useless, as there appears to be no gain in training loss compared to ordinary pretraining. Meanwhile, given a checkpoint from ordinary pretraining, continually pretraining with explicit memory exhibits a visible decrease in training loss. This comparison implies that the plain, implicit memory stage might be necessary for pretraining a Memory^3 model. One possible explanation for this phenomenon is that in the beginning of pretraining, the model is too weak to understand and leverage the explicit memories it generates. Then, to reduce distraction, the self-attention layers learn to always ignore these memories, thus hindering indefinitely the development of explicit memory.

Another modification is to reduce the cost of the explicit memory stage. Recall from Section 3.1 that each 64-token chunk attends to five explicit memories, or equivalently five 128-token references if using cold start, increasing the amount of tokens by 10 times. The inference process avoids the cost of memory encoding by precomputation or warm start, but during pretraining, the references need to be encoded in real time as the model parameters are constantly evolving. Our solution is to let the chunks share their references to reduce the total number of references in a batch. Specifically, each chunk of a training sequence retrieves only one reference, and in compensation, attends to the references of the previous four chunks, besides its own reference. Each train sequence has length 2048 and thus 32 chunks, so it is equipped with $32 \times 128 = 4096$ reference tokens. The hidden features of these reference tokens are discarded once passing the last memory layer, since after that they no longer participate in the update of the hidden feature of the train tokens. Hence, each step in the explicit memory stage takes slightly more than twice the amount of time of a step in the implicit memory stage.

It is necessary to avoid information leakage when equipping the training data with references (i.e. the train sequence and its retrieved references could be the same text), for otherwise training becomes too easy and the model would not learn much. Previously, Retro [14] requires that no train sequence can retrieve a reference from the same document, but this criterion may be insufficient since near-identical paragraphs may appear

in multiple documents. Thus, we require that no train sequence can be accompanied by a reference sequence that has greater than 90% overlap with it. The overlap is measured by the length of their longest common subsequence divided by the length of the reference. Specially, given any train sequence $\mathbf{t}$ and reference $\mathbf{r}$, define their overlap by

$$\text{overlap}(\mathbf{t},\mathbf{r}) := \frac{1}{|\mathbf{r}|}\max\big\{N \,\big|\, \exists\, 1 \le i_1 < \cdots < i_N \le |\mathbf{t}| \text{ and } \exists\, 1 \le j_1 < \cdots < j_N \le |\mathbf{r}| \\ \text{and } |i_N - i_1| \le 2|\mathbf{r}| \text{ such that } \mathbf{t}_{i_k} = \mathbf{r}_{j_k} \text{ for } k = 1,\ldots,N\big\}. \quad (3.3)$$

The constraint $|i_N - i_1| \le 2|\mathbf{r}|$ ensures that the overlap is not over-estimated as $|\mathbf{t}| \to \infty$.

4 Pretraining data

This section describes the procedures for collecting and filtering our pretraining dataset and knowledge base (or reference dataset).

4.1 Data collection

The pretrain data is gathered from English and Chinese text datasets, mostly publicly available collections of webpages and books. We also include code, SFT data (supervised finetuning), and synthetic data.

Specially, the English data mainly consists of RedPajamaV2 [106], SlimPajama [100] and the Piles [40], in total 200 TB prior to filtering. The Chinese data mainly comes from Wanjuan [48], Wenshu [119], and MNBVC [78], in total 500 TB prior to filtering. The code data mainly comes from Github, and we take the subset with the highest repository stars. The SFT data is included since these samples generally have higher quality than the webpages. We use the same data as in SFT training (Section 6.1), except that these samples are treated as ordinary texts during pretraining, i.e. all tokens participate in the loss computation, not just the answer tokens.

4.2 Filtering

The raw data is filtered with three steps: deduplication, rule-based filtering, and model-based filtering.

First, deduplication is performed with MinHash for most of the datasets. One exception is RedPajamaV2, which already comes with deduplication labels.

Second, we devise heuristic, rule-based filters analogous to the ones from [22, 72, 89]. The purpose is to eliminate texts that are ostensibly unsuitable for training, such as ones that only contain webpage source codes, random numbers, or incomprehensible shards. Our filters remove documents with less than 50 words, documents whose mean word lengths exceed 10 characters, documents with 70% of context being non-alphabetic characters, documents whose fractions of unique words are disproportionately high, documents whose entropy of unigrams is excessively low, and so on.

Finally, we select the subset of data with highest "quality", a score produced by a fine-tuned BERT model. Specially, we sample ten thousand documents and grade them by the XinYu-70B model [62,65] with prompt-guided generation. The prompt asks the model to determine whether the input text is informative and produce a score between 0 and 5. Then, these scores are used to finetune the Tiny-BERT model [54], which has only 14 M parameters. The hyperparameters of this finetuning are optimized with respect to a held-out validation set. After that, we use this lightweight BERT to grade the entire dataset.

Remark 4.1. Recall from Section 3.5 that the pretraining data of Memory^3 should emphasize abstract knowledges and minimize specific knowledges. The purpose is to not only obtain a lightweight LLM with an ideal distribution of knowledges in accordance with the memory hierarchy (Fig. 2.4), but also prevent the specific knowledges from hindering the learning process of the model. The focus of our prompt on "informativeness" might be contradictory to this goal, since the selected texts that are rich in information content may contain too many specific knowledges. For future versions of Memory^3, we will switch to a model-based filter favoring texts that exhibit more reasoning and less specifics.

The filtered dataset consists of around four trillion tokens, and its composition is illustrated in Fig. 4.1.

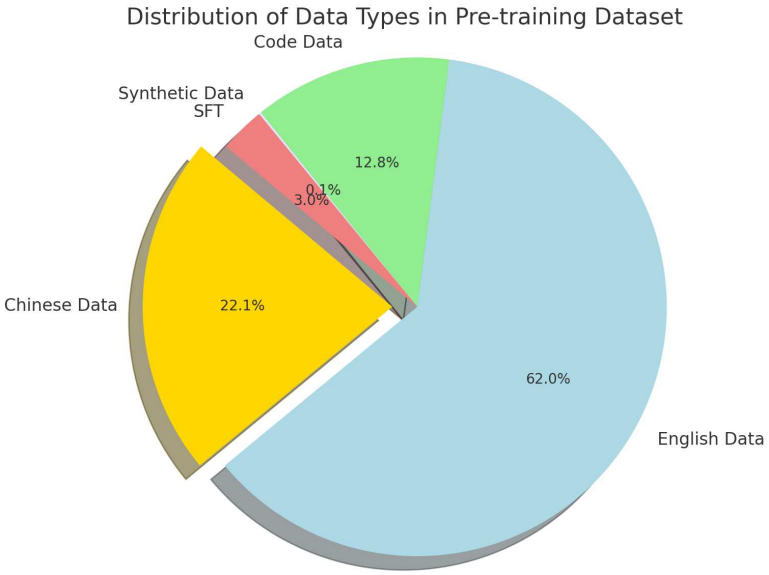

Figure 4.1: Composition of our pretraining dataset.

4.3 Tokenizer

Similar to our dataset, our tokenizer mainly consists of Chinese and English tokens. The English vocabulary comes from the 32000 tokens of the LLaMA2 tokenizer. We include roughly the same amount of Chinese tokens produced from byte-pair encoding (BPE).

The BPE is trained on a 20 GB Chinese corpus that consists of Chinese news and e-books. After deduplication, the final vocabulary has 60299 tokens.

4.4 Knowledge base

The knowledge base (or reference dataset) is used during training and inference as the source of explicit memories, as depicted in Fig. 1.2. It consists of reference texts that are split into token sequences with length less than or equal to 128, as described in Section 3.1.

Heuristically, a larger knowledge base is always better, as long as it does not contain misinformation, so it is not surprising that the reference dataset of Retro contains its entire pretrain dataset [14]. Nevertheless, the storage of explicit memories is more costly than plain texts despite our sparsification (Section 3.3), and thus to save storage space, we select a small subset of our pretrain dataset as the knowledge base.

With a focus on high quality data, we include for references the English Wikipedia, WikiHow, the Chinese baike dataset, the subset of English and Chinese books whose titles appear academic, Chinese news, synthetic data and high quality codes. These texts are tokenized and split into chunks of 128 tokens, resulting in 1.1×10^8 references in total.

One may be curious whether our knowledge base may contain some of the evaluation questions, rendering our evaluation results (Section 7.1) less credible. To prevent such leakage, we include in our evaluation code a filtering step, such that for each evaluation question, if a retrieved reference has an overlap with the question that exceeds a threshold, then it is discarded. This deduplication is analogous to the one used when preparing for the explicit memory pretraining stage (Section 3.6), with overlap measured by (3.3). The threshold $2/3$ is chosen since we observe that typically a reference that contains a question would have an overlap greater than or equal to 80%, while a relevant but distinct reference would have an overlap less than or equal to 40%.

Remark 4.2. Currently, the compilation of the knowledge base is based on human preference. For future versions of Memory^3, we plan to take a model-oriented approach and measure the fitness of a candidate reference by its actual utility, e.g. the expected decrease in the validation loss of the LLM conditioned on this reference being retrieved by a random validation sample.

5 Pretrain

This section describes the details of the pretraining process. The two-stage pretrain and memory-augmented data follow the designs introduced in Section 3.6. As an interpretation, the Memory^3 model during the implicit memory stage develops its reading comprehension, which is necessary during the explicit memory stage for initiating memory formation.

5.1 Set-up

Training is conducted with the Megatron-DeepSpeed package [76] and uses mixed-precision training with bfloat16 model parameters, bfloat16 activations, and float32 AdamW

states. The batch size is around 4 million training tokens with sequence length 2048, not including the reference tokens. The weight decay is the common choice of 0.1.

We adopt the "warmup-stable-decay" learning rate schedule of MiniCPM [51], which is reportedly better than the usual cosine schedule in term of training loss reduction. The learning rate linearly increases to the maximum value, then stays there for the majority of training steps, and finally in the last 10% steps decays rapidly to near zero. Our short-term experiments confirm the better performance of this schedule. Nevertheless, frequent loss spikes and loss divergences are encountered during the official pretraining, so we have to deviate from this schedule and manually decrease the learning rate to stabilize training.

Originally, it is planned that both the implicit/explicit memory stages go through the entire 4T token pretrain dataset (Section 4). Due to the irremediable loss divergences, both stages have to be terminated earlier.

5.2 Implicit memory stage

The training loss and learning rate schedule are plotted in Fig. 5.1. Whenever severe loss divergence occurs, we restart from the last checkpoint before the divergence with a smaller learning rate, and thus the divergences are not shown in the figure. Eventually, the training terminates at around 3.1 T tokens, when reducing the learning rate can no longer avoid loss divergence.

5.3 Explicit memory stage

The explicit memories enter into the Memory^3 model at this stage. The training steps are slower since the model needs to encode the references retrieved for the pretrain data to explicit memories in real time, and each step takes a bit more than twice the time of a step in the implicit memory stage. The training loss and learning rate schedule are plotted in Fig. 5.2.

The loss divergence soon becomes irremediable at around 120 B training tokens, much shorter than the planned 4 T tokens, and training has to stop there. One possible cause is that the model is initialized from the last checkpoint of the implicit memory stage, located immediately before the break down of that stage, and thus is already at the brink

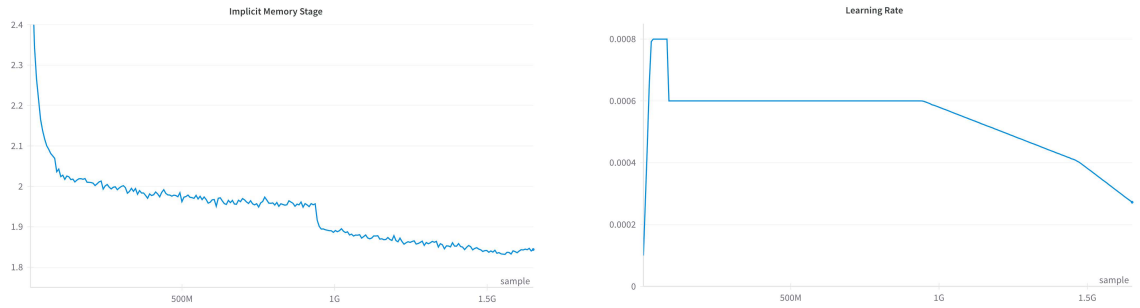

Figure 5.1: The implicit memory stage. Left: Training loss. Right: Learning rate schedule.

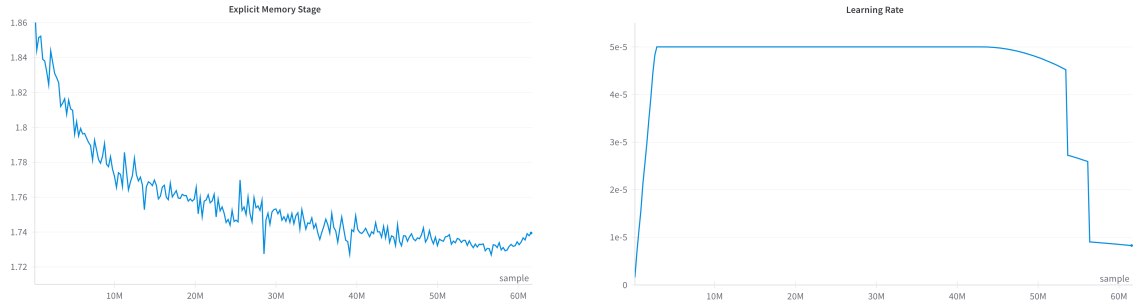

Figure 5.2: The explicit memory stage. Left: Training loss. Right: Learning rate schedule.

of divergence. The smaller learning rate of the explicit memory stage delays the onset of divergence but not for long.

6 Fine-tuning and alignment

This section describes our model finetuning, specifically supervised finetuning and direct preference optimization (DPO).

6.1 Supervised finetuning

Analogous to the StableLM model [12], our Memory^3 model is finetuned on a diverse collection of SFT datasets. We use the following datasets, which are publicly accessible on the Hugging Face Hub: UltraChat [31], WizardLM [124], SlimOrca [64], ShareGPT [112], Capybara [28], Deita [69], and MetaMathQA [129]. We also include synthetic data with emphasis on multi-round chat, mathematics, commonsense and knowledge. Each training sample consists of one or more rounds of question and answer pairs. We remove any sample with more than eight rounds. The final composition is listed in Table 6.1.

Table 6.1: Composition of SFT dataset.

Dataset	Source	Number of samples
UltraChat	HuggingFaceH4/ultrachat_200k	194409
WizardLM	WizardLM/WizardLM_evol_instruct_V2_196k	80662
SlimOrca	Open-Orca/SlimOrca-Dedup	143789
ShareGPT	openchat/openchat_sharegpt4_dataset	3509
Capybara	LDJnr/Capybara	7291
Deita	hkust-nlp/deita-10k-v0	2860
MetaMathQA	meta-math/MetaMathQA	394418
Multi-round Chat	synthetic	20000
Mathematics	synthetic	20000
Commonsense	synthetic	150000
Knowledge	synthetic	270000

The training process uses the cosine learning rate schedule with a max learning rate of 5×10^{-5} and a 10% linear warmup phase. The weight decay is 0.1, batch size is 512, and max sequence length is 2048 tokens. Finetuning is performed for 3 epochs.

6.2 Direct preference optimization

The Memory^3 model is further finetuned by DPO [90], to align with human preference and improve its conversation skills. The DPO dataset consists of general conversations (UltraFeedback Binarized [109]), math questions (Distilabel Math [7]) and codes questions (Synth Code [33]). The training uses the cosine learning rate schedule with max lr 4×10^{-6}. The inverse temperature β of the DPO loss is set to 0.01. The improvement from DPO is displayed in Section 7.2.

7 Evaluation

We evaluate the general abilities (benchmark tasks), conversation skills, professional abilities (law and medicine), and facutality & hallucination of the Memory^3 model. We also measure its decoding speed. Our model is compared with SOTA LLMs of similar and larger sizes, as well as RAG models.

7.1 General abilities

To evaluate the general abilities of Memory^3, we adopt all tasks from the Huggingface leaderboard and also include two Chinese tasks. Most of the results are displayed in Table 7.1, while TruthfulQA is listed in Table 7.4. All results are obtained in bfloat16 format, using the lm-evaluation-harness package [41] and the configuration of HuggingFace Open LLM leaderboard [11], i.e. the number of few-shot examples and grading methods.

As described in Section 4.4, to prevent cheating, a filtering step is included in the retrieval process so that the model cannot copy from references that resemble the evaluation questions.

The results of our model without using explicit memory is included, which indicates that explicit memory boosts the average score by 2.51%. In comparison, the score difference between Llama2-7B and 13 B is 4.91% while the latter has twice the amount of non-embedding parameters. Thus, it reasonable to say that explicit memory can increase the "effective model size" by $2.51/4.91 \approx 51.1\%$. (Also, the score difference between Qwen-1.8B and 4 B is 8.48% while the latter has 167% more non-embedding parameters. With respect to this scale, explicit memory increases the "effective model size" by $1.2.51/8.48 \times 1.67 \approx 49.4\%$.)

We also include the results of Memory^3 with vector compression (Section 3.3). Even though the key-value vectors of the explicit memories are compressed to 8.75% of their original sizes, the performance of our model does not show any degradation.

Other supplementary evaluations can be found in Appendix C.

Next, we compare with a LLM that is pretrained with text retrieval. Specially, we consider the largest version of the Retro++ model [111], Retro++ XXL with 9.5 B parameters.

All tasks from [111, Table 6] are taken, except for HANS, which is not available on lm-eval-harness, and all tasks are zero-shot. Similar to Table 7.1, Memory^3 is tested with a filtering threshold of 2/3. The results are listed in Table 7.2, where Memory^3 outperforms the model with much larger parameter size and reference dataset size.

Table 7.1: Few-shot evaluation of general abilities. The model sizes only include non-embedding parameters.

			English					Chinese	
LLM	Size	Avg.	ARC-C	HellaSwag	MMLU	Winogrande	GSM8k	CEVAL	CMMLU
Falcon-40B	41B	55.75	61.86	**85.28**	56.89	**81.29**	21.46	41.38	42.07
Llama2-7B-Chat	6.5B	46.87	52.90	78.55	48.32	71.74	7.35	34.84	34.40
Llama2-13B-Chat	13B	51.78	59.04	81.94	54.64	74.51	15.24	38.63	38.43
Llama3-8B-it	7.0B	**65.77**	**62.03**	78.89	**65.69**	75.77	**75.82**	50.52	51.70
Vicuna-13B-v1.5	13B	52.02	57.08	81.24	56.67	74.66	11.30	41.68	41.53
Mistral-7B-v0.1	7.0B	59.15	59.98	83.31	64.16	78.37	37.83	45.91	44.49
Gemma-2B-it	2.0B	36.64	38.02	40.36	55.74	35.29	55.88	8.26	29.94
Gemma-7B-it	7.8B	47.23	51.45	71.96	53.52	67.96	32.22	27.93	25.70
MiniCPM-2B-SFT	2.4B	54.37	47.53	71.95	51.32	67.72	45.26	48.07	48.76
Phi-2	2.5B	55.70	61.09	75.11	58.11	74.35	54.81	34.40	32.04
ChatGLM3-6B	5.7B	54.62	41.38	66.98	50.54	64.25	51.25	54.01	53.91
Baichuan2-7B-Chat	6.5B	55.16	52.73	74.06	52.77	69.77	28.28	53.12	55.38
Qwen1.5-1.8B-Chat	1.2B	49.67	38.74	60.02	45.87	59.67	33.59	55.57	54.22
Qwen1.5-4B-Chat	3.2B	58.15	43.26	69.73	55.55	64.96	52.24	61.89	59.39
Qwen1.5-7B-Chat	6.5B	64.80	56.48	79.02	60.52	66.38	54.36	**68.20**	**68.67**
Memory^3-SFT	2.4B	63.31	58.11	80.51	59.68	74.51	52.84	59.29	58.24
- vector compression	2.4B	63.33	57.94	80.65	59.66	75.14	52.24	59.66	58.05
- without memory	2.4B	60.80	57.42	73.14	57.29	74.35	51.33	56.32	55.72

Table 7.2: Zero-shot comparison of LLMs pretrained with retrieval. The scores of Retro++ are taken from [111]. The size of a reference dataset is its number of tokens. The non-embedding parameter size of Retro++ is inferred from its vocabulary size.

LLM	Param size	Avg.	HellaSwag	BoolQ	Lambada	RACE
Retro++ XXL	9.1B	61.0	70.6	70.7	72.7	43.2
Memory^3-SFT	2.4B	**64.7**	83.3	80.4	57.9	45.3
	Reference size		PiQA	Winogrand	ANLI-R2	WiC
	330B		77.4	65.8	35.5	52.4
	14.3B		76.6	75.8	41.6	56.9

7.2 Conversation skill

Next we evaluate the conversation skills of Memory^3. We use MT-Bench (the Multi-turn Benchmark) [132] that consists of multi-round and open-ended questions. The results

Table 7.3: MT-Bench scores. The model sizes only include non-embedding parameters.

LLM	Size	MT-Bench Score
Phi-3	3.6B	8.38
Mistral-7B-Instruct-v0.2	7.0B	7.60
Qwen1.5-7B-Chat	6.5B	7.60
Zephyr-7B-beta	7.0B	7.34
MiniCPM-2B-DPO	2.4B	6.89
Llama-2-70B-Chat	68B	6.86
Mistral-7B-Instruct-v0.1	7.0B	6.84
Llama-2-13B-Chat	13B	6.65
Llama-2-7B-Chat	6.5B	6.57
MPT-30B-Chat	30B	6.39
ChatGLM2-6B	6.1B	4.96
Falcon-40B-Instruct	41B	4.07
Vicuna-7B	6.5B	3.26
Memory^3-SFT	2.4B	5.31
Memory^3-DPO	2.4B	5.80

are listed in Table 7.3, including the Memory^3 model finetuned by DPO introduced in Section 6.2.

We obtain all these scores using GPT-4-0613 as grader, following the single answer grading mode of MT-Bench. Our model outperforms Vicuna-7B, Falcon-40B-Instruct, and ChatGLM2-6B with fewer parameters.

7.3 Hallucination and factuality

Despite considerable progress, LLMs still face issues with hallucination, leading to outputs that often stray from factual accuracy [94]. Conceptually, Memory^3 should be less vulnerable to hallucination, since its explicit memories directly correspond to reference texts, whereas compressing texts into the model parameters might incur information loss. To evaluate hallucination, we select two English datasets, TruthfulQA [66] and HaluEval, and one Chinese dataset [61], HalluQA [18]. TruthfulQA is implemented with lm-evaluation-harness [41], while HaluEval and HalluQA are implemented with UHGEval [65]. The results are shown in Table 7.4, with Memory^3 achieving the highest scores on most tasks.

7.4 Professional tasks

One benefit of using explicit memory is that the LLM can easily adapt to new fields and tasks by updating its knowledge base. One can simply import task-related references into the knowledge base of Memory^3, and optionally, convert them to explicit memories in the case of warm start. Then, the model can perform inference with this new knowledge, skipping the more costly and possibly lossy process of finetuning, and running faster than

Table 7.4: Evaluation of hallucination. HaluE and TruQA denote HaluEval and TruthfulQA, respectively. Bolded numbers are the best results. The model sizes only include non-embedding parameters. Vicuna-13B-v1.5 gets one N/A since that entry is near zero and seems abnormal.

			English				Chinese
LLM	Size	Avg.	HaluE-QA	HaluE-Dialogue	TruQA-MC1	TruQA-MC2	HalluQA
Falcon-40B	41B	35.37	46.84	40.80	27.29	41.71	20.18
Llama2-13B	13B	28.01	23.34	31.05	25.95	36.89	22.81
Vicuna-13B-v1.5	13B	37.07	24.93	37.35	35.13	50.88	N/A
Baichuan2-13B	13B	37.64	46.02	45.45	26.81	39.79	30.12
Gemma-7B	7.8B	37.03	50.91	48.19	20.69	46.65	18.71
Mistral-7B-v0.1	7.0B	34.18	40.68	37.64	28.03	42.60	21.93
Llama2-7B	6.5B	36.80	52.46	51.93	25.09	38.94	15.59
Baichuan2-7B	6.5B	38.63	**62.33**	47.84	23.01	37.46	22.51
ChatGLM3-6B	5.7B	40.96	43.38	50.03	33.17	49.87	28.36
Qwen1.5-4B-Chat	3.2B	33.30	24.64	37.72	29.38	44.74	30.00
Phi-2	2.5B	38.31	50.71	39.55	31.09	44.32	25.89
MiniCPM-SFT	2.4B	36.47	49.24	47.80	24.11	37.51	23.71
Gemma-2B	2.0B	38.04	53.41	52.22	24.60	39.78	20.18
Qwen1.5-1.8B-Chat	1.2B	37.52	47.18	52.11	26.68	40.57	21.05
Memory^3-SFT	2.4B	**48.60**	56.61	**53.91**	**38.80**	**57.72**	**35.96**

RAG. This cost reduction has been demonstrated in Fig. 1.1 and Appendix A, and could facilitate the rapid deployment of LLMs across various industries.

Besides cost reduction, we need to demonstrate that Memory^3 can perform no worse than RAG. We consider two professional tasks in law and medicine. The legal task consists of multiple-choice questions from the Chinese National Judicial Examination (JEC-QA) dataset [133]. The field-specific references are legal documents from the Chinese national laws and regulations database [105]. These references are merged with our general-purpose knowledge base (Section 4.4) for inference.

The medical task consists of the medicine-related questions of C-Eval, MMLU and CMMLU, specifically from the following subsets:

- C-Eval: clinical medicine, basic medicine.
- MMLU: clinical knowledge, anatomy, college medicine, college biology, nutrition, virology, medical genetics, professional medicine.
- CMMLU: anatomy, clinical knowledge, college medicine, genetics, nutrition, traditional Chinese medicine, virology.

Our knowledge base is supplemented with medical texts from the open-source medical books dataset [97].

The results are shown in Table 7.5, and Memory^3 achieves better performance than most of the models. All evaluations use 5-shot prompting. The RAG models retrieve from the same knowledge bases and FAISS indices, except that they receive text references

Table 7.5: Comparison with RAG on professional tasks.

	JEC-QA			MED		
LLM	3 refs	5 refs	7 refs	3 refs	5 refs	7 refs
MiniCPM-2B-SFT	38.83	37.65	37.94	53.73	53.29	52.84
Gemma-2B	28.16	28.06	25.29	42.04	42.49	42.96
Gemma-2B-it	30.04	31.13	29.34	41.70	43.24	42.66
Llama-2-7B	28.06	24.70	24.90	45.14	44.43	37.96
Llama-2-7B-Chat	26.18	25.10	25.20	48.18	47.29	39.39
Phi-2	25.00	25.30	23.32	50.05	45.42	45.59
Qwen1.5-1.8B-Chat	42.98	43.87	41.50	52.16	52.50	52.16
Qwen1.5-4B-Chat	51.98	50.49	50.99	61.19	61.02	61.06
Memory^3-2B-SFT	39.38			56.22		

instead of explicit memories. They only retrieve once for each question, using only the question text for query, so the 5-shot examples do not distract the retrieval. Since the optimal number of references is not known for these RAG models, we test them for $3, 5$, and 7 references per question, and it seems that $3 \sim 5$ references are optimal. The usual formatting for RAG is used, i.e. header 1 + reference 1 + reference 2 + reference 3 + header 2 + few-shot examples + question, all separated by line breaks.

The performance plotted in Fig. 1.3 (right) is the average of the scores of the two tasks in Table 7.5 with five references.

7.5 Inference speed

Finally, we evaluate the decoding speed or throughput of Memory^3, measured by generated tokens per second. The results are compared to those of RAG models, to quantify the speedup of explicit memory over text retrieval.

A direct comparison of speeds is uninformative: The memory hierarchy (Fig. 2.4) implies that the Memory^3 model is more reliant on retrieval to supply knowledge, and naturally Memory^3 performs retrieval with higher frequency (5 references per 64 tokens, possibly higher in future versions). Therefore, it is necessary to jointly compare performance and speed. The speed measured in this section is plotted against the retrieval-augmented test accuracy from Section 7.4, resulting in Fig. 1.3 (right).

We measure decoding speed on a A800 GPU, and run all models with Flash Attention [29]. All models receive an input of batch size 32 and length 128 tokens, and generate an output with length 128 tokens. The throughput is computed by 32×128 divided by the time spent. We test each model 9 times, remove the first record, and take the average of the rest. Memory^3 performs $2 \times 128/64 - 1 = 3$ retrievals (the -1 means that the first decoded chunk inherits the explicit memories retrieved by the last input chunk). Each retrieval uses 32 queries to get 32×5 explicit memories. We consider the warm start scenario, with the explicit memories precomputed and saved to drives. We implement the worst case scenario, such that the reference ids are reset to be unique after vector search and the memory cache on RAM is disabled, forcing Memory^3 to load 32×5 memories

from drives. Meanwhile, each RAG model performs one retrieval with query length 64 tokens, receives 5 references for each sample, and inserts them at the beginning of the sample, similar to the setup for Table 7.5.

The results are listed in Table 7.6 (local server). The throughput of these models without retrieval is also provided.

In addition, we study the throughput of these models when they are hosted on an end-side device and retrieve from a knowledge base on a remote server. Specifically, we use Jetson AGX Orin, and the server uses the vector engine MyScale [79]. The models are run with plain attention, with batch size 1. To simulate real-world use cases, the input is a fixed text prompt, with approximately 128 tokens, while the exact length can vary among different tokenizers. The output length is fixed to be 128 tokens. The results are listed in Table 7.6 (end-side device), and the Memory^3 model.

Remark 7.1. Table 7.6 indicates that our Memory^3-2B model is $1 - 733/1131 \approx 35.2\%$ slower than the same model without using memory. This is peculiar considering that reading explicit memories accounts for only a tiny fraction of the total compute

$$\frac{2.884 \times 10^{-3}\ \text{TFlops}}{1.264\ \text{TFlops}} \approx 0.228\%.$$

(The calculations are based on Appendix A.) Controlled experiments indicate that the time consumption is mainly due to two sources:

- Loading the memory key-values from drives to GPU: This overhead becomes prominent as Memory^3 retrieves with higher frequency.
- Python implementation of chunkwise attention: When encoding a prompt, since each chunk attends to a different set of explicit memories, we use a for loop over the chunks to compute their attentions.

Table 7.6: Inference throughput, measured by tokens per second.

		Local server		End-side device	
LLM	Size	with retrieval	w/o retrieval	with retrieval	w/o retrieval
MiniCPM-2B	2.4B	501.5	974.0	21.7	51.79
Gemma-2B-it	2.0B	1581	2056	22.0	29.23
Gemma-7B-it	7.8B	395.6	1008	9.5	18.61
Mistral-7B-Instruct-v0.1	7.0B	392.9	894.5	11.1	28.7
Llama-2-7B-Chat	6.5B	382.8	1005	10.0	23.19
Llama-2-13B-Chat	13B	241.1	632.5	2.5	5.44
Qwen1.5-1.8B-Chat	1.2B	908.2	1770	-	-
Qwen1.5-4B-Chat	3.2B	460.7	1002	22.3	53.39
Qwen1.5-7B-Chat	6.5B	365.8	894.5	-	-
Phi-2	2.5B	622.2	1544	-	-
Memory^3-2B	2.4B	733.0	1131	27.6	44.36

They dominate other sources such as computing query vectors by the embedding model and searching the vector index. We will try to optimize our code to reduce these overheads to be as close as possible to 0.228% of the total inference time, e.g. implement the chunkwise attention with a CUDA kernel.

8 Conclusion

The goal of this work is to reduce the cost of LLM training and inference, or equivalently, to construct a more efficient LLM that matches the performance of larger and slower LLMs. We analyze LLMs from the new perspective of knowledge manipulation, characterizing the cost of LLMs as the transport cost of "knowledges" in and out of various memory formats. Two causes of inefficiency are identified, namely the suboptimal placement of knowledges and the knowledge traversal problem. We solve both problems with explicit memory, a novel memory format, along with a new training scheme and architecture. Our preliminary experiment, the Memory^3-2B model, exhibits stronger abilities and higher speed than many SOTA models with greater sizes as well as RAG models.

For future work, we plan to explore the following directions:

1. Efficient training with abstract knowledges: Ideally, the training cost of Memory^3 model should be proportional to the small amount of non-separable knowledges, approaching the learning efficiency of humans. One approach is to filter the training data to maximize abstract knowledges and minimize specific knowledges (cf. Section 3.5 and Remark 4.1), and preferably the LLM should assess the quality of its own training data and ignore the unhelpful tokens.

2. Human-like capabilities: As described in the introduction, the explicit memory allows for interesting cognitive functions such as handling infinite contexts (conversion of working memory to explicit memory), memory consolidation (conversion of explicit memory to implicit memory), and conscious reasoning (reflection on the memory recall process). These designs may further improve the efficiency and reasoning ability of Memory^3.

3. Compact representation of explicit memory: The explicit memory of humans can be subdivided into episodic memory, which involve particular experiences, and semantic memory, which involve general truths [56]. This classification is analogous to our definition of specific and abstract knowledges. Our current implementation of explicit memory is closer to the episodic memory of humans, as each memory directly corresponds to a reference text. To improve its reasoning ability, one can try to equip Memory^3 with semantic memories, e.g. obtained from induction on the episodic memories.

Besides these broad topics, there are also plenty of engineering works that can be done. For instance, an internalized retrieval process that matches sparse attention queries with memory keys (Remark 3.1), sparser memory heads with routing (Remark 3.4), memory extraction that fully preserves contexts (Remark 3.2), compilation of the knowledge base

based on machine preference (Remark 4.2), reduction of the time consumption of explicit memory to be proportional to its compute overhead (Remark 7.1), and so on.

Appendix A Cost estimation

This section provides the calculations for Fig. 1.1, and we equate cost with the amount of compute measured in Tflops.

Our 2.4 B Memory[3] model is adopted as the backbone. Recall from Section 3.4 that this model has shape:

- Transformer blocks $L = 44$.
- Query heads $H = 40$ and key-value heads $H_{kv} = 8$.
- Head dimension $d_h = 80$ and hidden dimension $d = Hd_h = 3200$.
- MLP width $W = d$.
- Vocabulary size as well as LM head size $n_{\text{vocab}} = 60416$.
- Memory layers $L_{\text{mem}} = 22$, which is also the depth of the deepest memory layer.

Fix a separable knowledge $\mathcal{K}$, and represent it by one of its realizations $\mathbf{t}$ (Definition 2.5), and assume that $\mathbf{t}$ has length $l_{\text{ref}} = 128$ tokens, following the setup of our reference dataset (Section 4.4). Recall from Section 3.3 that each memory has $l_{\text{mem}} = 8$ tokens per memory head, and it is read by a chunk of length $l_{\text{chunk}} = 64$.

Since we want to show that explicit memory is cheaper than implicit memory and RAG, it suffices to use coarse lower bounds on their costs.

A.1 Implicit memory

The write cost of implicit memory or model parameters is the training compute with $\mathbf{t}$ as input. Usually the training data of Transformer LLMs have length $2048 \sim 8192$, so we assume that $\mathbf{t}$ is a subsequence of a train sample $\mathbf{t}_{\text{train}}$ with length $l_{\text{train}} = 2048$. By [81], the training compute of one step with one sample is approximately

$$3 \cdot 2 \cdot \left[L \left(l_{\text{train}} \left(2d^2 + 2dd_h H_{kv} + 3dW \right) + 2 \frac{l_{\text{train}}^2}{2} d \right) + l_{\text{train}} n_{\text{vocab}} d \right],$$

where 3 means that the backward step costs twice as the forward step (and thus 3 times in total), the first 2 means that the compute of matrix multiplication involves same amount of additions and multiplications. The five terms in the bracket come from QO embedding, KV embedding, MLP, attention, and LM head, respectively. The lower order terms, such as layer normalizations, are omitted. The fraction of the compute attributable to $\mathbf{t}$ is given by

$$3 \cdot 2 \cdot \left[L \left(l_{\text{ref}} \left(2d^2 + 2dd_h H_{kv} + 3dW \right) + 2 l_{\text{ref}} \frac{l_{\text{train}}}{2} d \right) + l_{\text{ref}} n_{\text{vocab}} d \right].$$

Assume that one training step is sufficient for storing knowledge $\mathcal{K}$ into model parameters. Then, the write cost is equal to the above term, and we obtain

$$\text{cost}_{\text{write}} \approx 2.24 \text{ TFlops}.$$

Meanwhile, we lower bound the read cost by zero

$$\text{cost}_{\text{read}} \geq 0 \text{ TFlops}.$$

This lower bound is obviously correct and suits our comparison, since it makes implicit memory appear more competitive. The difficulty in estimating the cost is that the correspondence between knowledges and parameters is not fully understood. Nevertheless, we describe a possible way to obtain a reasonable bound. Recall from Section 1 that the model parameters suffer from the issue of knowledge traversal such that each parameter (and thus each implicit memory) is invoked during each call of the LLM. So the read cost of each implicit memory does not depend on its usage count n_k, but instead on the total amount of model calls during the lifespan of this LLM. Dividing the total amount of inference compute used by this LLM by the amount of knowledges it possesses gives an estimation of the average read cost of a knowledge. The amount of knowledges in the LLM can be upper bounded based on the knowledge capacities measured by [4].

A.2 Explicit memory

The write cost of an each explicit memory mainly comes from L_{mem} self-attention layers, $L_{\text{mem}} - 1$ MLP layers, and L_{mem} token sparsification operations (computing the full attention matrix)

$$\begin{aligned}\text{cost}_{\text{write}} &= 2 \cdot \left[L_{\text{mem}} \left(l_{\text{ref}} \left(2d^2 + 2dd_h H_{kv} \right) + 2 \frac{l_{\text{ref}}^2}{2} d \right) + (L_{\text{mem}} - 1)(l_{\text{ref}} \cdot 3dW) + L_{\text{mem}} \left(l_{\text{ref}}^2 d \right) \right] \\ &\approx 0.308 \text{ TFlops}.\end{aligned}$$

The read cost consists of the attention to the sparse tokens of an explicit memory from the chunk that retrieves this memory

$$\text{cost}_{\text{read}} = 2L_{\text{mem}} \cdot 2 l_{\text{chunk}} l_{\text{mem}} d \approx 1.44 \times 10^{-4} \text{ TFlops}.$$

A.3 External information

The write cost of text retrieval-augmented generation is set to be zero, since the reference is stored as plain text

$$\text{cost}_{\text{write}} = 0 \text{ TFlops}.$$

The read cost is the additional compute brought by the retrieved references that are inserted in the prompt. To make RAG appear more competitive, we assume that only a chunk of the prompt or decoded text with length l_{chunk} can attend to the references, and each reference can only attend to itself, which in general is not true. Then,

$$\begin{aligned}\text{cost}_{\text{write}} &\geq 2 \cdot \left[L \left(l_{\text{ref}} \left(2d^2 + 2dd_h H_{kv} \right) + 2l_{\text{ref}} \left(\frac{l_{\text{ref}}}{2} + l_{\text{chunk}} \right) d \right) + (L-1)(l_{\text{ref}} \cdot 3dW) \right] \\ &\approx 0.624 \text{ TFlops.}\end{aligned}$$

In summary, the total cost (TFlops) of writing and reading each separable knowledge in terms of its expected usage count n is given by

$$\begin{aligned}c_{\text{implicit}}(n) &\geq 2.24, \\ c_{\text{explicit}}(n) &= 0.308 + 0.000144n, \\ c_{\text{external}}(n) &\geq 0.624n.\end{aligned}$$

These curves are plotted in Fig. 1.1. Hence, if $n \in (0.494, 13400)$, then it is optimal to store the knowledge as an explicit memory.

Remark A.1 (Knowledge Retention)**.** One aspect not covered by problem (1.1) is the retention of knowledges in the model if its parameters are updated, e.g. due to finetuning. Both implicit memory and explicit memory are vulnerable to parameter change. Usually, model finetuning would include some amount of pretrain data to prevent catastrophic forgetting [84]. Similarly, if some explicit memories have already been produced, then they need to be rebuilt in order to remain readable by the updated model. It is an interesting research direction to design a more efficient architecture such that the implicit and explicit memories are robust with respect to model updates.

Appendix B Vector compression

Regarding the vector quantizer discussed in Sections 3.3 and 7.1, we use the composite index of FAISS with index type OPQ20x80-Residual2x14-PQ8x10. It can encode a 80-dimensional bfloat16 vector into a 14-dimensional uint8 vector, and thus its compression rate is $(80 \times 2)/(14 \times 1) \approx 11.4$.

To train this quantizer, we sample references from our knowledge base, encode them into explicit memories by our Memory^3-2B-SFT model, and feed these key-value vectors to the quantizer. The references are sampled uniformly and independently, so the training is not biased towards the references that are retrieved by any specific evaluation task.

Appendix C Supplementary evaluation results

First, Table C.1 records the growth of the test scores (Table 7.1) over the two pretraining stages and SFT. We believe that for future versions of Memory^3, fixing the loss divergence during the implicit memory stage will allow the explicit memory stage to proceed much further (cf. Section 5.3), and thus increase the performance boost of the second stage.

Next, recall that for the evaluations in Section 7.1, a filter is included in the retrieval process to prevent copying, which removes references that overlap too much with the evaluation question. The filtering threshold should lie between 100% and the usual level

of overlap between two related but distinct texts, and we set it to 2/3 in Table 7.1. Table C.2 records the impact of the filtering threshold on the test scores. The scores are stable for most tasks, indicating that their questions do not appear in our knowledge basis.

Finally, Table C.3 studies the influence of the few-shot prompts on the benchmark tasks. Recall that the number of few-shot examples for each task is ARC-C (25), HellaSwag (10), MMLU (5), Winogrande (5), GSM8k (5) as in HuggingFace OpenLLM Leaderboard [11], and we also adopt CEVAL (5), CMMLU (5). Interestingly, the boost from explicit memory increases from 2.51% to 3.70% as we switch to 0-shot.

Table C.1: Performance of Memory^3-2B at different stages of training. The setup of the evaluation tasks is the same as in Table 7.1.

		English					Chinese	
LLM	Avg.	ARC-C	HellaSwag	MMLU	Winogrande	GSM8k	CEVAL	CMMLU
Implicit memory stage	42.13	40.27	64.57	41.62	61.96	5.23	40.12	41.17
Explicit memory stage	45.12	42.66	79.21	41.81	59.43	6.29	42.20	44.21
- without memory	42.89	42.15	66.98	39.79	61.80	6.44	39.97	43.13
SFT	63.31	58.11	80.51	59.68	74.51	52.84	59.29	58.24
- without memory	60.80	57.42	73.14	57.29	74.35	51.33	56.32	55.72

Table C.2: Influence of the filtering threshold on the test scores in Table 7.1.

Threshold	Avg.	ARC-C	HellaSwag	MMLU	Winogrande	GSM8k	CEVAL	CMMLU
no filter	63.71	58.11	83.37	59.65	74.51	52.84	59.29	58.22
80%	63.62	58.11	82.69	59.65	74.51	52.84	59.29	58.24
2/3	63.31	58.11	80.51	59.68	74.51	52.84	59.29	58.24
without memory	60.80	57.42	73.14	57.29	74.35	51.33	56.32	55.72

Table C.3: Few-shot versus 0-shot for the benchmark tasks in Table 7.1.

Mode	Avg.	ARC-C	HellaSwag	MMLU	Winogrande	GSM8k	CEVAL	CMMLU
Few-shot	63.31	58.11	80.51	59.68	74.51	52.84	59.29	58.24
- without memory	60.80	57.42	73.14	57.29	74.35	51.33	56.32	55.72
0-shot	58.23	58.79	83.29	60.53	75.85	13.50	57.95	57.74
- without memory	54.54	57.34	73.15	58.59	74.98	10.46	54.53	54.26

Acknowledgments

We thank Prof. Zhiqin Xu, Prof. Zhouhan Lin, Fangrui Liu, Liangkai Hang, Ziyang Tao, Xiaoxing Wang, Mingze Wang, Yongqi Jin, Haotian He, Guanhua Huang, Yirong Hu for helpful discussions.

This work is supported by the NSFC Major Research Plan – Interpretable and General Purpose Next-generation Artificial Intelligence of China (Grant No. 92270001).

References

[1] M. Abdin et al., Phi-3 technical report: A highly capable language model locally on your phone, *arXiv: 2404.14219*, 2024.
[2] J. Achiam et al., GPT-4 technical report, *arXiv:2303.08774*, 2023.
[3] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, GQA: Training generalized multi-query transformer models from multi-head checkpoints, *arXiv:2305.13245*, 2023.
[4] Z. Allen-Zhu and Y. Li, Physics of language models: Part 3.3, knowledge capacity scaling laws, *arXiv: 2404.05405*, 2024.
[5] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, É. Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, The falcon series of open language models, *arXiv:2311.16867*, 2023.
[6] Anthropic AI, The Claude 3 model family: Opus, Sonnet, Haiku, *Claude-3 Model Card* `https://www.anthropic.com/claude`, 2024.
[7] Argilla, Distilabel Math Preference DPO. `https://huggingface.co/datasets/argilla/distilabel-math-preference-dpo`, 2023.
[8] Azure AI Services. GPT-4 and GPT-4 turbo models. `https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models#gpt-4-and-gpt-4-turbo-models`, 2024. [Accessed 22-05-2024]
[9] J. Bai et al., Qwen technical report, *arXiv:2309.16609*, 2023.
[10] P. J. Bayley and L. R. Squire, Failure to acquire new semantic knowledge in patients with large medial temporal lobe lesions, *Hippocampus*, 15(2):273–280, 2005.
[11] E. Beeching, C. Fourrier, N. Habib, S. Han, N. Lambert, N. Rajani, O. Sanseviero, L. Tunstall, and T. Wolf, Open LLM leaderboard, `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`, 2023.
[12] M. Bellagente et al., Stable lm 2 1.6 b technical report, *arXiv:2402.17834*, 2024.
[13] A. Bertsch, U. Alon, G. Neubig, and M. Gormley, Unlimiformer: Long-range transformers with unlimited length input, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[14] S. Borgeaud et al., Improving language models by retrieving from trillions of tokens, in: *International conference on machine learning*, PMLR, 2206–2240, 2022.
[15] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, *arXiv:2402.03216*, 2024.
[16] Y. Chen, T. Guan, and C. Wang, Approximate nearest neighbor search by residual vector quantization, *Sensors*, 10(12):11259–11273, 2010.
[17] Z.-A. Chen, Y. Li, T. Luo, Z. Zhou, and Z.-Q. J. Xu, Phase diagram of initial condensation for two-layer neural networks, *CSIAM Trans. Appl. Math.*, 5(3):448–514, 2024.
[18] Q. Cheng, T. Sun, W. Zhang, S. Wang, X. Liu, M. Zhang, J. He, M. Huang, Z. Yin, K. Chen, and X. Qiu, Evaluating hallucinations in Chinese large language models, *arXiv:2310.03368*, 2023.
[19] A. Chowdhery et al., Palm: Scaling language modeling with pathways, *arXiv:2204.02311*, 2022.
[20] B. Chughtai, A. Cooney, and N. Nanda, Summing up the facts: Additive mechanisms behind factual recall in LLMs, *arXiv:2402.07321*, 2024.
[21] A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso, Towards automated circuit discovery for mechanistic interpretability, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 16318–16352, 2023.
[22] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, Unsupervised cross-lingual representation learning at scale, *arXiv:1911.02116*, 2019.
[23] S. Corkin, What's new with the amnesic patient H.M.? *Nat. Rev. Neurosci.*, 3(2):153–160, 2002.

[24] N. Cowan, The magical number 4 in short-term memory: A reconsideration of mental storage capacity, *Behav. Brain. Sci.*, 24(1):87–114, 2001.
[25] N. Cowan, *Working memory capacity: Classic Edition*, Routledge, 2016.
[26] D. Dai, L. Dong, Y. Hao, Z. Sui, B. Chang, and F. Wei, Knowledge neurons in pretrained transformers, *arXiv:2104.08696*, 2021.
[27] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, Transformer-XL: Attentive language models beyond a fixed-length context, *arXiv:1901.02860*, 2019.
[28] L. Daniele and Suphavadeeprasit, Amplify-instruct: Synthetically generated diverse multi-turn conversations for effecient LLM training, `https://huggingface.co/datasets/LDJnr/Capybara`, 2023.
[29] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, Flashattention: Fast and memory-efficient exact attention with IO-awareness, *Adv. Neural Inf. Process. Syst.*, Vol. 35, 16344–16359, 2022.
[30] N. Dey, G. Gosal, Zhiming, Chen, H. Khachane, W. Marshall, R. Pathria, M. Tom, and J. Hestness, Cerebras-GPT: Open compute-optimal language models trained on the cerebras wafer-scale cluster, *arXiv:2304.03208*, 2023.
[31] N. Ding, Y. Chen, B. Xu, Y. Qin, Z. Zheng, S. Hu, Z. Liu, M. Sun, and B. Zhou, Enhancing chat language models by scaling high-quality instructional conversations, *arXiv:2305.14233*, 2023.
[32] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, The faiss library, *arXiv:2401.08281*, 2024.
[33] P. V. Duy, synth_code_preference_4k, `https://huggingface.co/datasets/pvduy/synth_code_ preference_4k`, 2023.
[34] M. Elbayad, J. Gu, E. Grave, and M. Auli, Depth-adaptive transformer, *arXiv:1910.10073*, 2020.
[35] N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah, Toy models of superposition, *Transformer Circuits Thread*, 2022. `https://transformer-circuits.pub/2022/toy_model/index.html`.
[36] J. Fang, L. Tang, H. Bi, Y. Qin, S. Sun, Z. Li, H. Li, Y. Li, X. Cong, Y. Yan, X. Shi, S. Song, Y. Lin, Z. Liu, and M. Sun, UniMem: Towards a unified view of long-context large language models, *arXiv:2402.03009*, 2024.
[37] W. Fedus, B. Zoph, and N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022.
[38] E. Frantar and D. Alistarh, QMoE: Practical sub-1-bit compression of trillion-parameter models, *arXiv:2310.16795*, 2023.
[39] J. D. Gabrieli, N. J. Cohen, and S. Corkin, The impaired learning of semantic knowledge following bilateral medial temporal-lobe resection, *Brain Cogn.*, 7(2):157–177, 1988.
[40] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, The pile: An 800 gb dataset of diverse text for language modeling, *arXiv:2101.00027*, 2021.
[41] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, A framework for few-shot language model evaluation, `https://zenodo.org/records/10256836`, 2023.
[42] M. Geva, J. Bastings, K. Filippova, and A. Globerson, Dissecting recall of factual associations in auto-regressive language models, *arXiv:2304.14767*, 2023.
[43] M. Geva, R. Schuster, J. Berant, and O. Levy, Transformer feed-forward layers are key-value memories, *arXiv:2012.14913*, 2021.
[44] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256, 2010.
[45] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. D. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, and Y. Li, Textbooks are all you need, *arXiv:2306.11644*, 2023.
[46] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, Retrieval augmented language model pre-training, in: *Proceedings of the 37th International Conference on Machine Learning*, PMLR, 3929–3938, 2020.

[47] Y. Hao, D. Angluin, and R. Frank, Formal language recognition by hard attention transformers: Perspectives from circuit complexity, *Trans. Assoc. Comput.*, 10:800–810, 2022.
[48] C. He, Z. Jin, C. Xu, J. Qiu, B. Wang, W. Li, H. Yan, J. Wang, and D. Lin, Wanjuan: A comprehensive multimodal dataset for advancing english and chinese large models, *arXiv:2308.10755*, 2023.
[49] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: *Proceedings of the IEEE International Conference on Computer Vision*, IEEE, 1026–1034, 2015.
[50] J. Hoffmann et al., Training compute-optimal large language models, *arXiv:2203.15556*, 2022.
[51] S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao, X. Zhang, Z. L. Thai, K. Zhang, C. Wang, Y. Yao, C. Zhao, J. Zhou, J. Cai, Z. Zhai, N. Ding, C. Jia, G. Zeng, D. Li, Z. Liu, and M. Sun, MiniCPM: Unveiling the potential of small language models with scalable training strategies, *arXiv:2404.06395*, 2024.
[52] Y. Huang, S. Hu, X. Han, Z. Liu, and M. Sun, Unified view of grokking, double descent and emergent abilities: A perspective from circuits competition, *arXiv:2402.15175*, 2024.
[53] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, Mistral 7b, *arXiv:2310.06825*, 2023.
[54] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, Tinybert: Distilling bert for natural language understanding, *arXiv:1909.10351*, 2020.
[55] J. Kaddour, The minipile challenge for data-efficient language models, *arXiv:2304.08442*, 2023.
[56] E. Kandel, J. Koester, S. Mack, and S. Siegelbaum, *Principles of Neural Science*, McGraw Hill LLC, 2021.
[57] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, Scaling laws for neural language models, *arXiv:2001.08361*, 2020.
[58] O. Khattab and M. Zaharia, Colbert: Efficient and effective passage search via contextualized late interaction over bert, *arXiv:2004.12832*, 2020.
[59] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, Reducing activation recomputation in large transformer models, *arXiv:2205.05198*, 2022.
[60] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, Efficient memory management for large language model serving with pagedattention, *arXiv:2309.06180*, 2023.
[61] J. Li, X. Cheng, X. Zhao, J.-Y. Nie, and J.-R. Wen, Halueval: A large-scale hallucination evaluation benchmark for large language models, in: *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.
[62] M. Li, M.-B. Chen, B. Tang, S. Hou, P. Wang, H. Deng, Z. Li, F. Xiong, K. Mao, P. Cheng, and Y. Luo, Newsbench: A systematic evaluation framework for assessing editorial capabilities of large language models in chinese journalism, *arXiv:2403.00862*, 2024.
[63] Y. Li, S. Bubeck, R. Eldan, A. D. Giorno, S. Gunasekar, and Y. T. Lee, Textbooks are all you need II: phi-1.5 technical report, *arXiv:2309.05463*, 2023.
[64] W. Lian, G. Wang, B. Goodson, E. Pentland, A. Cook, C. Vong, and "Teknium", Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification, `https://huggingface.co/Open-Orca/SlimOrca`, 2023.
[65] X. Liang, S. Song, S. Niu, Z. Li, F. Xiong, B. Tang, Y. Wang, D. He, P. Cheng, Z. Wang, and H. Deng, Uhgeval: Benchmarking the hallucination of chinese large language models via unconstrained generation, *arXiv:2311.15296*, 2024.
[66] S. Lin, J. Hilton, and O. Evans, TruthfulQA: Measuring how models mimic human falsehoods, in: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 3214–3252, 2022.
[67] H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma, Sophia: A scalable stochastic second-order optimizer for language model pre-training, *arXiv:2305.14342*, 2024.
[68] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, Lost in the middle: How language models use long contexts, *arXiv:2307.03172*, 2023.
[69] W. Liu, W. Zeng, K. He, Y. Jiang, and J. He, What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning, *arXiv:2312.15685*, 2023.

[70] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava, Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time, *arXiv: 2305.17118*, 2023.
[71] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Ré, and B. Chen, Deja Vu: Contextual sparsity for efficient llms at inference time, in: *Proceedings of Machine Learning Research*, PMLR, 202:22137–22176, 2023.
[72] S. Longpre et al., A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity, *arXiv:2305.13169*, 2023.
[73] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, Estimating the carbon footprint of bloom, a 176 b parameter language model, *arXiv:2211.02001*, 2022.
[74] T. Luo, Z.-Q. J. Xu, Z. Ma, and Y. Zhang, Phase diagram for two-layer relu neural networks at infinite-width limit, *J. Mach. Learn. Res.*, 22(71):1–47, 2021.
[75] A. Lv, Y. Chen, K. Zhang, Y. Wang, L. Liu, J.-R. Wen, J. Xie, and R. Yan, Interpreting key mechanisms of factual recall in transformer-based language models, *arXiv:2403.19521*, 2024.
[76] Megatron-DeepSpeed. `https://github.com/microsoft/Megatron-DeepSpeed`, 2022.
[77] W. Merrill and A. Sabharwal, A logic for expressing log-precision transformers, *arXiv:2210.02671*, 2023.
[78] MOP-LIWU Community and MNBVC Team. Mnbvc: Massive never-ending bt vast chinese corpus. `https://github.com/esbatmop/MNBVC`, 2023.
[79] MyScale. MyScaleDB. `https://github.com/myscale/MyScaleDB`. [Accessed 20-03-2024]
[80] R. Nakano et al., WebGPT: Browser-assisted question-answering with human feedback, *arXiv: 2112.09332*, 2021.
[81] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, Efficient large-scale language model training on GPU clusters using megatron-LM, *arXiv:2104.04473*, 2021.
[82] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah, In-context learning and induction heads, *arXiv:2209.11895*, 2022.
[83] OpenAI. GPT-4 turbo and GPT-4. `https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4`, 2024. [Accessed 22-05-2024].
[84] L. Ouyang et al., Training language models to follow instructions with human feedback, *Adv. Neural Inf. Process. Syst.*, Vol. 35, 27730–27744, 2022.
[85] A. Pearce, A. Ghandeharioun, N. Hussein, N. Thain, M. Wattenberg, and L. Dixon. Do machine learning models memorize or generalize? *People+ AI Research*, `https://pair.withgoogle.com/explorables/grokking/`, 2023.
[86] B. Peng et al., RWKV: Reinventing RNNs for the transformer era, *arXiv:2305.13048*, 2023.
[87] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, Grokking: Generalization beyond overfitting on small algorithmic datasets, *arXiv:2201.02177*, 2022.
[88] P. Qi, X. Wan, G. Huang, and M. Lin, Zero bubble pipeline parallelism, *arXiv:2401.10241*, 2023.
[89] J. W. Rae et al., Scaling language models: Methods, analysis & insights from training Gopher, *arXiv: 2112.11446*, 2021.
[90] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, Direct preference optimization: Your language model is secretly a reward model, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[91] D. Raposo, S. Ritter, B. Richards, T. Lillicrap, P. C. Humphreys, and A. Santoro, Mixture-of-depths: Dynamically allocating compute in transformer-based language models, *arXiv:2404.02258.*, 2024.
[92] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters, in: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 3505–3506, 2020.
[93] N. Ratner, Y. Levine, Y. Belinkov, O. Ram, I. Magar, O. Abend, E. Karpas, A. Shashua, K. Leyton-Brown, and Y. Shoham, Parallel context windows for large language models, *arXiv:2212.10947*, 2022.
[94] V. Rawte, S. Chakraborty, A. Pathak, A. Sarkar, S. T. I. Tonmoy, A. Chadha, A. Sheth, and A. Das, The troubling emergence of hallucination in large language models – an extensive definition, quantification,

and prescriptive remediations, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2541–2573, 2023.
[95] Y. Ruan, C. J. Maddison, and T. Hashimoto, Observational scaling laws and the predictability of language model performance, *arXiv:2405.10938*, 2024.
[96] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, Toolformer: Language models can teach themselves to use tools, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[97] Scienceasdf. Medical books. `https://github.com/scienceasdf/medical-books`. [Accessed 20-03-2024].
[98] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism, *arXiv:1909.08053*, 2020.
[99] Snowflake AI Research. Snowflake arctic: The best LLM for enterprise AI – efficiently intelligent, truly open, `https://www.snowflake.com/blog/arctic-open-efficient-foundation-language-models-snowflake/`, 2024. [Accessed: 2024-05-15]
[100] D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, and N. Dey, SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. `https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`, 2023.
[101] A. Srivastava et al., Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, *arXiv:2206.04615*, 2023.
[102] A. Stolfo, Y. Belinkov, and M. Sachan, A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 7035–7052, 2023.
[103] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, Roformer: Enhanced transformer with rotary position embedding, *Neurocomputing*, 568:127063, 2024.
[104] S. Sukhbaatar, E. Grave, G. Lample, H. Jegou, and A. Joulin, Augmenting self-attention with persistent memory, *arXiv:1907.01470*, 2019.
[105] The Chinese National Laws and Regulations Database, `https://flk.npc.gov.cn/`. [Accessed 20-03-2024]
[106] Together Computer. RedPajama: An open dataset for training large language models, `https://github.com/togethercomputer/RedPajama-Data`, 2023.
[107] Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei, You only cache once: Decoder-decoder architectures for language models, *arXiv:2405.05254*, 2024.
[108] H. Touvron et al., Llama 2: Open foundation and fine-tuned chat models, *arXiv:2307.09288*, 2023.
[109] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib, N. Sarrazin, O. Sanseviero, A. M. Rush, and T. Wolf, Zephyr: Direct distillation of LM alignment, *arXiv:2310.16944*, 2023.
[110] S. Tworkowski, K. Staniszewski, M. Pacek, Y. Wu, H. Michalewski, and P. Miłoś, Focused transformer: Contrastive training for context scaling, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[111] B. Wang, W. Ping, P. Xu, L. McAfee, Z. Liu, M. Shoeybi, Y. Dong, O. Kuchaiev, B. Li, C. Xiao, A. Anandkumar, and B. Catanzaro, Shall we pretrain autoregressive language models with retrieval? A comprehensive study, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 7763–7786, 2023.
[112] G. Wang, S. Cheng, X. Zhan, X. Li, S. Song, and Y. Liu, Openchat: Advancing open-source language models with mixed-quality data, *arXiv:2309.11235*, 2023.
[113] K. Wang, A. Variengien, A. Conmy, B. Shlegeris, and J. Steinhardt, Interpretability in the wild: A circuit for indirect object identification in GPT-2 small, *arXiv:2211.00593*, 2022.
[114] L. Wang, L. Li, D. Dai, D. Chen, H. Zhou, F. Meng, J. Zhou, and X. Sun, Label words are anchors: An information flow perspective for understanding in-context learning, *arXiv:2305.14160*, 2023.
[115] M. Wang, H. He, J. Wang, Z. Wang, G. Huang, F. Xiong, Z. Li, W. E, and L. Wu, Improving generalization and convergence by enhancing implicit regularization, *arXiv:2405.20763*, 2024.
[116] W. Wang, L. Dong, H. Cheng, X. Liu, X. Yan, J. Gao, and F. Wei, Augmenting language models with long-term memory, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.

[117] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, Emergent abilities of large language models, *arXiv:2206.07682*, 2022.
[118] G. Weiss, Y. Goldberg, and E. Yahav, Thinking like transformers, *arXiv:2106.06981*, 2021.
[119] Wenshu, `https://wenshu.court.gov.cn/`. [Accessed 20-03-2024]
[120] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, Sustainable AI: Environmental implications, challenges and opportunities, *arXiv:2111.00364*, 2022.
[121] W. Wu, Y. Wang, G. Xiao, H. Peng, and Y. Fu, Retrieval head mechanistically explains long-context factuality, *arXiv:2404.15574*, 2024.
[122] Y. Wu, M. N. Rabe, D. Hutchins, and C. Szegedy, Memorizing transformers, in: *International Conference on Learning Representations*, ICLR, 2021.
[123] X. Xie, P. Zhou, H. Li, Z. Lin, and S. Yan, Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models, *arXiv:2208.06677*, 2023.
[124] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang, Wizardlm: Empowering large language models to follow complex instructions, *arXiv:2304.12244*, 2023.
[125] A. Yang et al., Baichuan 2: Open large-scale language models, *arXiv:2309.10305*, 2023.
[126] G. Yang, E. J. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao, Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer, *arXiv:2203.03466*, 2022.
[127] H. Yang, A mathematical framework for learning probability distributions, *J. Mach. Learn.*, 1(4):373–431, 2022.
[128] Y. Yao, N. Zhang, Z. Xi, M. Wang, Z. Xu, S. Deng, and H. Chen, Knowledge circuits in pretrained transformers, *arXiv:2405.17969*, 2024.
[129] L. Yu, W. Jiang, H. Shi, J. Yu, Z. Liu, Y. Zhang, J. T. Kwok, Z. Li, A. Weller, and W. Liu, Metamath: Bootstrap your own mathematical questions for large language models, *arXiv:2309.12284*, 2023.
[130] Z. Zhang, P. Lin, Z. Wang, Y. Zhang, and Z.-Q. J. Xu, Initialization is critical to whether transformers fit composite functions by inference or memorizing, *arXiv:2405.05409*, 2024.
[131] Z. Zhang et al., H2O: Heavy-hitter oracle for efficient generative inference of large language models, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[132] L. Zheng et al., Judging LLM-as-a-judge with MT-bench and Chatbot Arena, *Adv. Neural Inf. Process. Syst.*, Vol. 36, 2024.
[133] H. Zhong, C. Xiao, C. Tu, T. Zhang, Z. Liu, and M. Sun, JEC-QA: A legal-domain question answering dataset, *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9701–9708, 2020.