MemTensor

Metis: Memory Foundation Model

Zeyu Zhang[1,2*], **Ziliang Guo**[1*], **Yihang Sun**[1,4*], **Xichong Zhang**[1*], **Xixuan Hao**[1], **Zehao Lin**[1], **Yang Zhang**[3], **Xiaoyan Zhao**[3], **Tong Shen**[1,4], **Bo Tang**[1], **Zhi-Qin John Xu**[4], **Junchi Yan**[4], **Haofen Wang**[5], **Xu Chen**[2†], **Feiyu Xiong**[1], **Zhiyu Li**[1†], **Tat-Seng Chua**[3]

[1]**MemTensor (Shanghai) Technology Co., Ltd.,** [2]**Renmin University of China,** [3]**National University of Singapore,** [4]**Shanghai Jiao Tong University,** [5]**Tongji University**

arXiv:2607.26760v2 [cs.CL] 4 Aug 2026

Abstract

Recent advances in AI agents have increasingly internalized native capabilities into their underlying foundation models, giving rise to multimodal foundation models and large reasoning models. However, agent memory is still primarily implemented through external modules, leaving the native memory capability largely unexplored. In this paper, we take a first step toward this direction by introducing **memory foundation models**, which empower foundation models with native memory capabilities. We formalize **native memory** from two perspectives: a persistent and dynamically evolving memory state within the backbone, and native memory procedures that autonomously store and utilize information through model computation. We show that native memory offers advantages in architecture, end-to-end optimization, and efficiency. Based on this formulation, we propose **Metis**, the first prototype of memory foundation models. Metis introduces a new architecture that equips a foundation model with a native memory state, allowing historical information to be compressed into the model and accessed through memory attention. We construct large-scale memory-specific training data and introduce multiple optimization objectives to acquire these native memory procedures through mid-training. The online memory maintenance of Metis is gradient-free, and the memory update requires only a forward pass. At inference time, all learned model weights remain frozen, while the native memory states are autonomously transformed through standard forward computation. Through extensive experiments, we show that Metis exhibits native memory capabilities and further provide a detailed analysis of its strengths, limitations, and behaviors. To facilitate future research on memory foundation models, we release our project and model checkpoints.

Date: August 5, 2026
Correspondence: lizy@memtensor.cn, xu.chen@ruc.edu.cn
Author Legend: *Co-first authors, †Corresponding authors
Code: https://github.com/MemTensor/Metis
Model: https://huggingface.co/collections/IAAR-Shanghai/metis

1 Introduction

In recent years, large foundation models have achieved rapid development, demonstrating significant performance across many aspects, such as language modeling [1, 2], code generation [3–5], and complex reasoning [6–8]. This provides a solid foundation for constructing AI agents, which enables them to handle more complex tasks. Beyond the reasoning capabilities of foundation models, memory is another critical capability of AI agents, responsible for retaining past information and leveraging it to support future inference [9, 10]. In most previous works, memory is implemented

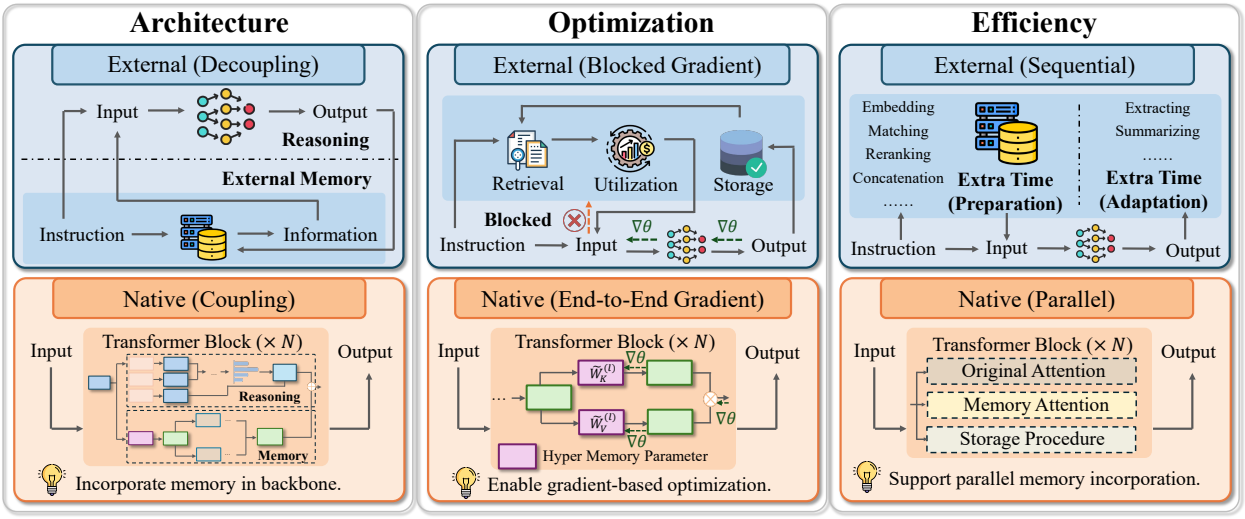

Figure 1 From external memory to native memory.

by a module external to foundation models, rather than being natively integrated into their architectures [11–14]. Representative approaches use Retrieval-Augmented Generation (RAG) to retrieve relevant textual information and incorporate it into the prompt to facilitate inference [11].

However, external memory suffers from several limitations presented in **Figure 1**. First, external memory is decoupled from backbones with separated targets and processing stages [11, 13]. External memory typically aims to construct an informative context as input, and backbones only perform conditional language modeling over the constructed context. Therefore, external memory may not provide the most useful information to support the backbone inference, and the backbone may not utilize the memory optimally. Second, end-to-end optimization is difficult for external memory because gradients cannot be effectively propagated through discrete memory operations [15]. As a result, performing domain-specific post-training becomes highly challenging. Although some RL-based strategies [16, 17] can partially alleviate this issue by optimizing memory operations with reward signals, they suffer from efficiency issues. Finally, external memory requires additional explicit operations over the storage outside backbones, which inevitably increases the online inference latency [9].

To address the limitations of external memory, we introduce **memory foundation models** that empower large foundation models with **native memory**. It converts memory from an external module into an internal mechanism of backbones, directly involved in forward computation. Specifically, memory foundation models can generate responses based on the input instructions and their native memory, with autonomous memory transformation. We define the native memory from two critical aspects:

- **Native Memory State.** Unlike traditional large foundation models, memory foundation models are natively stateful across multiple inferences, which can formulate, maintain, and utilize memory states inside backbones from prior inferences. Their memory states are dynamically represented as part of the parameters of backbones, whose semantic spaces are aligned at the pre-training or mid-training stage.
- **Native Memory Procedure.** Unlike memory engineering, memory foundation models natively integrate memory procedures within their inferences. Specific memory operations, such as remembering, forgetting, and updating, are accomplished autonomously alongside the backbone's forward computation, which impacts the native memory state based on input instructions.

Memory foundation models aim to internalize memory capability into the model's forward computation. The memory state can be represented as dynamic parameters of backbones, and memory procedures are executed through computation. Therefore, like general foundation models, memory foundation models can be optimized and adapted to specific domains in a data-driven manner through post-training. This transformation is similar to the evolution from large foundation models to large reasoning models [8], where Chain-of-Thought (CoT) [7] is natively integrated into inference to improve performance and efficiency. In addition, because native memory procedures can be integrated into the model's computation, they provide a foundation for improving the parallel efficiency of memory processing.

In this paper, we implement the first prototype of memory foundation models, named **Metis**. We design a new model architecture that has a native memory state inspired by Fast Weight Programming (FWP) [18], which can be integrated into the backbone computation through memory attention. Specifically, we propose the Metis blocks as the basic units for native memory. Each of them primarily consists of a hyper memory block and a local memory block. In addition, we empower Metis with native memory procedures by designing specific optimization objectives, including memory reconstruction and memory operation objectives. These two objectives correspond to the compression upper limit of memory states and operation targets. We also design a regularization objective to improve robustness in complex scenarios. To support this training, we synthesize large-scale memory-specific data from publicly available datasets, enabling Metis to acquire native memory capabilities through mid-training. Finally, we conduct extensive experiments to demonstrate the effectiveness of our proposed framework, and explore more aspects for analysis.

From a general perspective, a fundamental problem of memory results from the time-streaming property of online information. At the storage stage, memory systems cannot determine how the received information will be used in the future. At the inference stage, the original information is no longer accessible, and only the stored information can be utilized. Therefore, memory can be considered as a prediction problem, where the model predicts how received information will be utilized in the future. Like other prediction tasks in machine learning, memory capability can also be acquired at the pre-training stage and generalized to other domains, and memory foundation models can provide the architectural and optimization foundation.

Despite their promising properties, implementing memory foundation models remains highly challenging because their final goal is to completely eliminate the reliance on external memory in contexts. While Metis achieves great performance in memory-related tasks, it still faces several limitations. First, its performance degrades on long-term tasks, due to the information loss when compressed into fixed-size parameters. Second, it exhibits information confusion in some cases, possibly caused by the blending of semantics within the latent space. Despite these limitations, Metis provides a potential pathway to achieve memory foundation models. To benefit both the research community and industry, we release our project at https://github.com/MemTensor/Metis.

Our contributions are summarized as follows:
• We introduce memory foundation models and native memory with formal definitions, providing further analysis from the perspective of native memory state and native memory procedure.
• We propose the first prototype of memory foundation models, named Metis, which is implemented with novel memory architectures and optimization tasks.
• We conduct extensive experiments to verify the effectiveness of our model, followed by detailed studies from multiple perspectives. We also publicly release our project to benefit the research community and industry.

The rest of our paper is organized as follows. **Section 2** provides the formal definition of memory foundation models. **Section 3** details the model architecture of Metis. After that, we introduce our data construction pipeline in **Section 4** and outline the optimization in **Section 5**. Extensive experimental results and analysis are presented in **Section 6**. Finally, we review related work in **Section 7** and conclude in **Section 8**.

2 Memory Foundation Model

In this section, we provide a formal definition of the memory foundation model. Then, we introduce native memory from the perspectives of the native memory state and procedure. After that, we compare memory foundation models with previous works. Finally, we further discuss memory foundation models from the perspectives of lifelong learning and the evolving trends of AI agents.

2.1 Definition

We define the memory foundation model under the multi-step scenario. Let a continuous interaction process be formulated as a sequence of discrete time steps $t \in \{1, 2, \ldots, T\}$. At each step t, the foundation model receives an input instruction sequence denoted as X_t and generates a response sequence Y_t.

For traditional foundation models without memory, the generation relies entirely on the current input context. The autoregressive decoding of the k-th token in the response is typically expressed as $y_{t,k} \sim P(y \mid X_t, Y_{t,<k}; \theta)$, where $Y_{t,<k}$ denotes the previously generated tokens at step t, and θ represents the fixed parameters of the backbone. For

foundation models with external memory, the autoregressive decoding process is then conditioned on the context C_t alongside the input instruction. It can be expressed by $y_{t,k} \sim P(y \mid C_t, X_t, Y_{t,<k}; \theta)$, where C_t can be obtained from prior information $\{(X_i, Y_i)\}_{i=1}^{t-1}$. The external memory framework commonly has two explicit procedures, including the storage procedure $\mathbf{C}_t = \mathbf{C}_{t-1} \oplus \{(X_t, Y_t)\}$, and the retrieval procedure $C_t = \mathbf{C}_{t-1} \otimes X_t$. Here, $\oplus$ denotes the general writing operation, and $\otimes$ represents the general reading operation with the textual memory storage $\mathbf{C}_{t-1}$. Both of them are executed outside the model inference process.

Definition 1 (Memory Foundation Model)**.** The memory foundation model is defined as an autoregressive foundation model empowered by **native memory** across multi-step interactions. At each step t, the generation of the k-th token is conditioned on the input instruction X_t and the previously generated tokens $Y_{t,<k}$ by

$$y_{t,k} \sim P(y \mid X_t, Y_{t,<k}; \theta_t),$$

where the model parameter θ_t integrates information from previous steps into its native parametric space (*i.e.,* native memory state). Concurrently, θ_{t+1} is autonomously transformed during the forward computation inside the model based on the input instruction X_t and output Y_t (*i.e.,* native memory procedure).

In contrast, the memory foundation model internalizes memory into the backbone's computation, which is empowered with native memory. Instead of relying on an external explicit storage $\mathbf{C}_t$ and context C_t, it maintains a native memory state, which acts as dynamic parameters across multiple steps. In addition, rather than explicitly executing memory procedures outside backbones, memory procedures in the memory foundation model occur autonomously alongside the model's forward computation, such as operations like remembering, forgetting, and updating.

2.2 Native Memory State

In this paper, we adopt a strict definition for the source of memory. We only consider the information acquired during online interactions as memory, where information available before the interaction starts is excluded. In fact, such offline information is better viewed as knowledge rather than memory, because it does not contain trajectory-specific information for personalization and does not require real-time adaptation.

Since the stored information varies across different steps, the parameters θ_t cannot remain completely static. Consequently, at least a portion of the parameters must change dynamically according to the input, and we denote this dynamic part as the memory state $\mathbf{M}_t$. In the memory foundation model, the representation of stored information is supposed to be coupled with the backbone to participate in forward computation. Therefore, the native memory state should be represented in parametric form. Although textual memory offers advantages in interpretability and cross-model compatibility, its discrete representation results in low information density and requires repetitive prefilling. In contrast, parametric memory represents prior information in a dense form, which increases the efficiency of storage and utilization.

In addition, the semantic spaces of both dynamic parameters $\mathbf{M}_t$ and static parameters $\Phi = \theta_t \setminus \mathbf{M}_t$ must be aligned during the pre-training or mid-training stage before conducting online inference. This alignment enables the dynamic parameters at different steps to compute collaboratively with the static parameters. During online interactions, the native memory state can be updated and utilized through the native memory procedure, which empowers the memory foundation model with statefulness across different steps.

2.3 Native Memory Procedure

In terms of memory, storage and utilization are two core procedures to handle online information with the time-streaming property. Memory storage retains past information, while memory utilization leverages this stored information to support model inference. They aim to address the temporal mismatch between information supply and usage.

The memory storage procedure typically involves several specific operations, such as remembering, forgetting, and updating. From the perspective of foundation models, an input instruction contains both the intent and the content of information processing. For instance, *"Alice is 24 years old"* implies remembering her age, while *"Bob moved from London to Boston"* indicates updating his location. A native storage procedure should directly map the input instruction to the update value of the memory state. In contrast, external memory relies on rule-based and predefined operations to handle its intent and content separately. Although most operations can be categorized into insertion, deletion, and modification, the semantic intent and content cannot be easily decoupled into discrete rules. In fact, the storage procedure

essentially predicts how current information will be used in the future. Because rule-based procedures operate in a discrete function space, they struggle to achieve optimal prediction performance.

The primary goal of the memory utilization procedure is to assist inference with the stored information. From the foundation model perspective, it can be considered as letting the required information of the input instruction participate in the forward computation. For example, answering *"Where does Bob live now?"* requires previously stored living information to facilitate inference. Thus, a native memory utilization procedure should directly map the input instruction and memory state to the generated output. External memory designs rules to trigger retrieval, reranking, and concatenation. However, the information requirement cannot be defined and captured by discrete and finite rules. For example, an instruction may require information based on semantic similarity, emotion, or even complex combinations of implicit metrics. The memory utilization procedure predicts the information requirements, which is coupled with the inference process. Therefore, it should not be divided into discrete stages limited by discrete function spaces.

Consequently, the memory procedure should be modeled within a continuous function space and implemented via numerical computation, which is tightly coupled with the forward computation of the backbone. In a memory foundation model, the native memory procedure autonomously executes both memory storage and utilization during the forward computation. This native memory procedure should be established during the pre-training or mid-training stage. In addition, this memory procedure paradigm has significant advantages in both efficiency and end-to-end optimization.

2.4 Comparison with Previous Works

Test-time Training. Memory foundation models differ from test-time training (TTT) in three key aspects. First, TTT typically adapts the model within a single sequence, where the dynamic parameters are updated to better fit the current input. In contrast, memory foundation models are defined under multi-step interactions. Their dynamic parameters serve as persistent native memory states that store information from previous steps and support future inference.

Second, TTT does not explicitly provide native memory procedures. Its update is usually driven by self-supervised language modeling, which helps the model absorb prior information within the current sequence. However, it does not specify how the model should semantically remember, forget, update, or reflect on information according to input instructions. In contrast, memory foundation models are trained with memory reconstruction and operation objectives, enabling the model to autonomously execute semantic memory operations in the latent parametric space and transform the native memory state accordingly.

Third, many TTT methods are motivated by efficient long-context modeling, and they often introduce recurrent layers to replace full attention. However, memory foundation models pursue a different goal. They do not aim to replace the standard full-attention computation within the current step. Instead, they introduce information from previous interaction steps through native memory states as residuals.

In summary, TTT is primarily a mechanism for inference-time adaptation, while memory foundation models formulate memory as a persistent, instruction-driven, and procedure-aware capability of foundation models.

Memory-Augmented Neural Networks. Memory-augmented neural networks (MANNs) introduce additional memory modules to neural models, such as differentiable memory slots and learned read-write operations [19]. These models show that neural networks can store external information and retrieve it for later computation. Nevertheless, memory foundation models differ in how memory is integrated with the backbone. In MANNs, the memory module is usually a separate storage component controlled by a neural controller. Although the operations can be differentiable, the memory is still external to the main model parameters, which are often designed independently from the backbone.

In contrast, memory foundation models internalize memory into the backbone computation. The memory state is represented in a parametric form and participates directly in forward computation. The memory procedure is also modeled by the same continuous function space as the backbone, rather than being implemented as a separate controller over explicit slots. Therefore, memory foundation models can be regarded as a step from externally augmented memory toward native memory inside foundation models.

Other Methods. Compared with In-place TTT [20], MemGen [21] and δ-Mem [22], which still rely on textual memories in the context and use additional latent summaries to improve inference, memory foundation models remove textual memory from the context entirely. Compared with MEMO [23] and MemFT [24], which primarily handle offline documents, memory foundation models focus on test-time information. Compared with Memory^3 [25], which takes an

important step beyond textual RAG by encoding knowledge into retrievable explicit memories, memory foundation models further extend this direction toward native memory. While Memory^3 primarily constructs explicit memories from offline corpora and retrieves them to augment attention computation, memory foundation models internalize information acquired from online interactions as persistent dynamic states within the backbone. They further enable these states to be autonomously maintained and transformed through native memory procedures across multiple interaction steps.

2.5 Discussion

For memory foundation models, the onset of interaction represents a key transition from static to dynamic knowledge acquisition. Knowledge acquired before interaction originates from offline pre-training and is retained in static parameters, while information received during interaction is acquired at test time and stored in dynamic memory states. Therefore, θ_1 can also serve as initial supplementary information outside of pre-training. Because θ_1 captures transferable domain knowledge, models deployed in similar domains can be initialized with the same θ_1 to provide baseline information.

Moreover, native memory aligns with the evolving trend of foundation models. Inspired by large reasoning models [8], we find that an agent's external capabilities can be expressed natively by the foundation model through optimization. In other words, supervised data of target behaviors can activate internal capabilities and generalize them to other tasks. These native capabilities can provide advantages in generalization, efficiency, and optimization properties. Memory is also a critical agent capability that traditionally relies on external modules. Therefore, we argue that memory can also be natively triggered through memory-specific tasks. However, unlike reasoning, which is purely a process, memory also involves a storage entity. It requires us to modify the model architecture to incorporate a storage entity as the memory state. Then, both the memory state and procedures are supposed to be modeled under a collaborative function space. This enables us to empower foundation models with memory capabilities via an optimization-driven approach.

3 Metis Architecture

In this section, we first present some preliminaries. Then, we introduce Metis as the prototype of memory foundation models with the Metis block. After that, we demonstrate how this architecture supports native memory storage and utilization procedures through computation. Finally, we provide theoretical insights, theoretical error analysis, and further discussions. The overview of the Metis framework is presented in **Figure 2**.

3.1 Preliminaries

We adopt causal language models as the primary implementation of memory foundation models, as they are dominantly used in modern foundation models. We present the standard architecture of causal language models, which primarily consists of N stacked Transformer blocks followed by a language modeling head.

Transformer Block. To highlight the core architecture, we focus on causal self-attention and the feed-forward network (FFN), which are major components of modern Transformers. We omit other details, such as positional embeddings, multi-head attention strategies, and hybrid attention mechanisms, as they can be directly incorporated into our framework.

We denote the input of the l-th Transformer block as $\mathbf{H}^{(l-1)} \in \mathbb{R}^{L\times d}$, where L is the sequence length and d is the dimension of hidden states. After applying the pre-normalization function PreNorm, we obtain $\tilde{\mathbf{H}}^{(l)} = \text{PreNorm}(\mathbf{H}^{(l-1)})$, as the input of causal self-attention. We denote $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)} \in \mathbb{R}^{d\times d_k}, \mathbf{W}_V^{(l)} \in \mathbb{R}^{d\times d_v}$ as query, key, and value projection matrices of this layer. Then, we obtain the query state, key state, and value state of the l-th layer by

$$\mathbf{Q}^{(l)} = \tilde{\mathbf{H}}^{(l)}\mathbf{W}_Q^{(l)}, \quad \mathbf{K}^{(l)} = \tilde{\mathbf{H}}^{(l)}\mathbf{W}_K^{(l)}, \quad \mathbf{V}^{(l)} = \tilde{\mathbf{H}}^{(l)}\mathbf{W}_V^{(l)}.$$

After that, the causal self-attention can be calculated by

$$\mathbf{A}^{(l)} = \text{Softmax}\left(\frac{\mathbf{Q}^{(l)}(\mathbf{K}^{(l)})^\top}{\sqrt{d_k}} + \text{Mask}(L)\right)\mathbf{V}^{(l)}, \tag{1}$$

where d_k is the attention head dimension, and $\text{Mask}(L)$ is the causal mask defined as $\text{Mask}(L)_{i,j} = -\infty$ if $j > i$, and 0 otherwise. Then, it adds the projected output attention to the residual after projection by

$$\mathbf{H}'^{(l)} = \mathbf{H}^{(l-1)} + \mathbf{A}^{(l)}\mathbf{W}_O^{(l)},$$

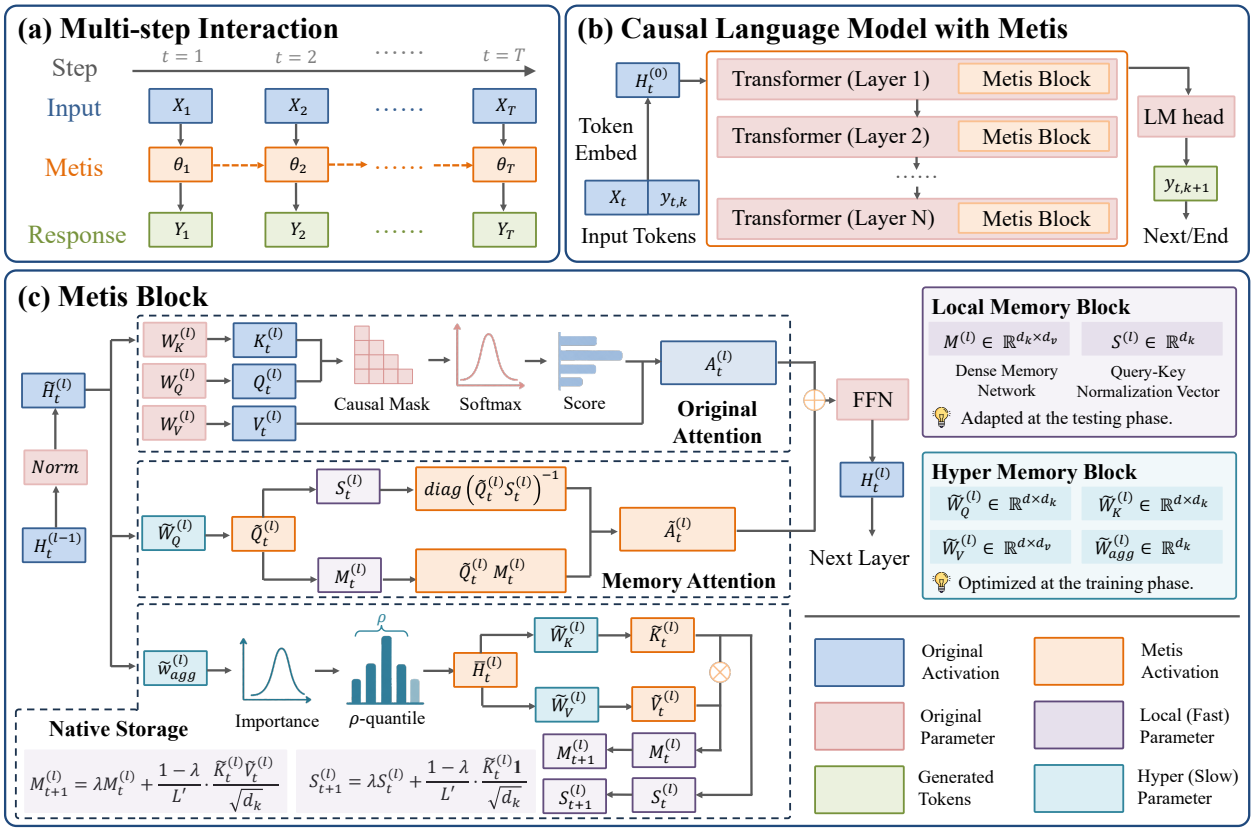

Figure 2 Overview of the Metis architecture.

where $\mathbf{W}_O^{(l)} \in \mathbb{R}^{d_v \times d}$. Finally, $\mathbf{H}'^{(l)}$ is passed through FFN with a residual connection to get the l-th layer output by

$$\mathbf{H}^{(l)} = \mathbf{H}'^{(l)} + \mathrm{FFN}(\mathrm{Norm}(\mathbf{H}'^{(l)})),$$

where the activation $\mathbf{H}^{(l)}$ is also the input of $(l+1)$-th layer.

Causal Language Model. We denote the sequence of input tokens as $X = (x_1, x_2, \ldots, x_L)$. A causal language model first maps these discrete tokens into continuous vector representations. Let $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ denote the token embedding matrix, where $|\mathcal{V}|$ is the vocabulary size. The initial hidden state $\mathbf{H}^{(0)} \in \mathbb{R}^{L \times d}$ is obtained by extracting the corresponding embeddings and combining them with positional embeddings. After that, this initial representation is processed sequentially through the stack of N Transformer blocks by

$$\mathbf{H}^{(l)} = \mathrm{TransformerBlock}^{(l)}(\mathbf{H}^{(l-1)}), \quad \text{for } l = 1, 2, \ldots, N.$$

Then, the final hidden state $\mathbf{H}^{(N)}$ represents the contextualized input, and the language modeling head maps this final state back to the vocabulary space to predict the probability distribution for the next token. This process is commonly modeled by a linear projection after normalization, followed by a softmax function

$$P(\cdot \mid x_{\le i}) = \mathrm{Softmax}(\mathrm{Norm}(\mathbf{H}_{(i)}^{(N)})\mathbf{W}_{\mathrm{LM}}),$$

where $\mathbf{H}_{(i)}^{(N)} \in \mathbb{R}^d$ is the final hidden vector at position i, and $\mathbf{W}_{\mathrm{LM}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ represents the projection matrix. After sampling $x_{i+1} \sim P(x_{i+1} \mid x_{\le i})$, this new token is appended to the sequence, and the model repeats the process until it decodes an end-of-sequence token or reaches the maximum length.

During the pre-training phase, the causal language model is optimized using the standard autoregressive next-token prediction objective. It minimizes the negative log-likelihood of the training sequences by

$$\theta = \arg\min_{\theta} \sum_{X \in \mathcal{D}} \sum_{i=1}^{|X|-1} -\log P(x_{i+1} \mid x_{\le i}; \theta),$$

where $\mathcal{D}$ represents the pre-training corpus and θ encompasses all the trainable parameters of the model.

3.2 Native Memory State

To implement the native memory state, we propose the Metis blocks inside Transformer blocks in **Figure 2(b)**, where each Metis block consists of a local memory block and a hyper memory block in **Figure 2(c)**. The local memory blocks are responsible for maintaining the dense representation of prior information, while the hyper memory blocks construct parametric function spaces for native memory procedures to transform memory states.

Local Memory Block. Local memory blocks maintain the memory state at the current step, so we define a dense memory network $\mathbf{M}^{(l)} \in \mathbb{R}^{d_k \times d_v}$ inside the l-th local memory block. At the step t, we denote it as $\mathbf{M}_t^{(l)}$. The model also maintains a query-key normalization vector as $\mathbf{S}_t^{(l)} \in \mathbb{R}^{d_k}$. Both $\mathbf{M}^{(l)}$ and $\mathbf{S}_t^{(l)}$ are dynamic parameters updated across different steps. Specifically, we set $\mathbf{M}_1^{(l)} = \mathbf{0}$ and $\mathbf{S}_1^{(l)} = \mathbf{0}$ by default.

Hyper Memory Block. Hyper memory blocks are responsible for updating the dynamic parameters in local memory blocks based on the intermediate activations of the current input X_t and output Y_t. Each of them consists of static parameters obtained through mid-training, which remain unchanged during interactions. It serves as the parametric foundation of the native memory storage procedure. Specifically, each hyper memory block is parameterized by several optimizable parameters. First, it has a learnable importance vector $\tilde{\mathbf{w}}_{\text{agg}}^{(l)} \in \mathbb{R}^d$, which scores the intermediate activations for adaptive aggregation. In addition, we set the memory key and value projection matrices $\tilde{\mathbf{W}}_K^{(l)} \in \mathbb{R}^{d \times d_k}$ and $\tilde{\mathbf{W}}_V^{(l)} \in \mathbb{R}^{d \times d_v}$, which map the selected hidden states into the memory keys and values for the local memory. We also set the memory query projection matrix $\tilde{\mathbf{W}}_Q^{(l)} \in \mathbb{R}^{d \times d_k}$ to reduce the error in the native memory utilization procedure.

3.3 Native Memory Procedure

The native memory procedure consists of memory storage and utilization procedures, as we discuss in **Section 2.3**. In the native memory storage procedure of our framework, hyper memory blocks update local memory blocks as part of the model computation, based on intermediate activations. In the native memory utilization procedure, local memory blocks incorporate the current memory states into the forward computation.

Native Memory Storage Procedure. After completing step t, we denote the input hidden states at the l-th layer as $\mathbf{H}_t^{(l-1)}$. Then, the hyper memory block aggregates it into a compact representation through a learned adaptive aggregation. Specifically, we first pre-normalize the hidden states as $\tilde{\mathbf{H}}_t^{(l)} = \text{PreNorm}(\mathbf{H}_t^{(l-1)})$ and score each of the L tokens with a learnable importance vector $\tilde{\mathbf{w}}_{\text{agg}}^{(l)} \in \mathbb{R}^d$, obtaining an importance distribution

$$\mathbf{p}_t^{(l)} = \text{Softmax}\left(\frac{\tilde{\mathbf{H}}_t^{(l)} \tilde{\mathbf{w}}_{\text{agg}}^{(l)}}{\tau}\right) \in \mathbb{R}^L,$$

where τ is a temperature coefficient. Then, we obtain a subset of positions based on the top-ρ. We sort these probabilities in descending order as $p_{(1)} \geq p_{(2)} \geq \cdots \geq p_{(L)}$, and keep the smallest prefix whose cumulative value reaches the threshold ρ. The number of selected positions can be expressed by

$$L_t' = \text{clip}\left(\min\left\{k : \sum_{r=1}^{k} p_{(r)} \geq \rho\right\},\ K_{\min},\ L\right),$$

where $K_{\min}$ denotes the minimum number of selected positions. These L_t' positions with the highest scores form the selected set $\mathcal{S}_t^{(l)}$, and we gather corresponding hidden states by

$$\bar{\mathbf{H}}_t^{(l)} = \mathbf{\Pi}_t^{(l)} \tilde{\mathbf{H}}_t^{(l)} \in \mathbb{R}^{L_t' \times d}, \quad \text{with } L_t' \ll L,$$

where $\mathbf{\Pi}_t^{(l)} \in \{0,1\}^{L_t' \times L}$, whose rows are the one-hot indicators of $\mathcal{S}_t^{(l)}$. Since the top-ρ selection is non-differentiable, we adopt a straight-through estimator that routes the gradients through the dense distribution $\mathbf{p}_t^{(l)}$, making the scorer $\tilde{\mathbf{w}}_{\text{agg}}^{(l)}$ end-to-end trainable. After that, we compute the projected memory key states and memory value states as

$$\tilde{\mathbf{K}}_t^{(l)} = \bar{\mathbf{H}}_t^{(l)} \tilde{\mathbf{W}}_K^{(l)}, \quad \tilde{\mathbf{V}}_t^{(l)} = \bar{\mathbf{H}}_t^{(l)} \tilde{\mathbf{W}}_V^{(l)}.$$

Finally, the dense memory network is updated based on $\tilde{\mathbf{K}}_t^{(l)}$ and $\tilde{\mathbf{V}}_t^{(l)}$ by

$$\mathbf{M}_{t+1}^{(l)} = \lambda \mathbf{M}_t^{(l)} + \frac{(1-\lambda)}{L_t'} \cdot \frac{\tilde{\mathbf{K}}_t^{(l)\top}}{\sqrt{d_k}} \tilde{\mathbf{V}}_t^{(l)}, \tag{2}$$

where λ represents the discount factor. In addition, the query-key normalization vector can be updated by

$$\mathbf{S}_{t+1}^{(l)} = \lambda \mathbf{S}_t^{(l)} + \frac{(1-\lambda)}{L_t'} \cdot \frac{\tilde{\mathbf{K}}_t^{(l)\top} \mathbf{1}}{\sqrt{d_k}}. \tag{3}$$

This native storage procedure is presented in **Figure 2(c)**. Based on the constructed function space, we aim to internalize various memory operations into the model's computation through optimization. Specifically, the selection and projection provide the model with compression capabilities. Meanwhile, semantic-based computation enables memory instructions to be understood and applied within the latent space. In practice, we find that replacing the linear update with a Gated Delta Network (GDN)-based [26] update obtains better performance, so Metis finally adopts the GDN-based update (GDU) strategy. **Section 6.3** and **Appendix C.1** compares the two implementations through ablation studies.

Native Memory Utilization Procedure. We define the memory attention as

$$\tilde{\mathbf{A}}_t^{(l)} = \mathrm{diag}\left(\tilde{\mathbf{Q}}_t^{(l)} \mathbf{S}_t^{(l)}\right)^{-1} \tilde{\mathbf{Q}}_t^{(l)} \mathbf{M}_t^{(l)}, \tag{4}$$

where $\tilde{\mathbf{Q}}_t^{(l)} = \tilde{\mathbf{H}}_t^{(l)} \tilde{\mathbf{W}}_Q^{(l)}$ denotes the memory query states with optimizable parameter $\tilde{\mathbf{W}}_Q^{(l)} \in \mathbb{R}^{d \times d_k}$. In practice, we add an identity vector to the normalization denominator to prevent numerical overflow and improve numerical stability. Then, the memory attention is integrated into the main branch of attention, and replaces **Equation (1)** with

$$\mathbf{A}_t^{(l)} = \gamma \cdot \mathrm{Softmax}\left(\frac{\mathbf{Q}_t^{(l)} (\mathbf{K}_t^{(l)})^\top}{\sqrt{d_k}} + \mathrm{Mask}(L)\right) \mathbf{V}_t^{(l)} + (1-\gamma) \cdot \mathrm{Norm}\left(\tilde{\mathbf{A}}_t^{(l)}\right), \tag{5}$$

where $\mathrm{Norm}(\cdot)$ is applied to the memory readout to align its scale with the original attention branch, $\mathbf{K}_t^{(l)}, \mathbf{V}_t^{(l)}$ are input key states and value states at the current step, and $\gamma \in [0, 1]$ balances the two branches.

3.4 Theoretical Insight of Native Memory Procedures

We provide theoretical insights on how information from previous steps influences subsequent inference through Metis blocks. At step t, we prepend an additional virtual memory prefix $\mathbf{P}_t^{(l)} \in \mathbb{R}^{L_p \times d}$ to the input $\tilde{\mathbf{H}}_t^{(l)}$ of the l-th attention layer, resulting in the augmented input

$$\hat{\mathbf{H}}_t^{(l)} = \begin{bmatrix} \mathbf{P}_t^{(l)} \\ \tilde{\mathbf{H}}_t^{(l)} \end{bmatrix}.$$

Then, we compute the corresponding query state $\hat{\mathbf{Q}}_t^{(l)}$ as follows

$$\hat{\mathbf{Q}}_t^{(l)} = \hat{\mathbf{H}}_t^{(l)} \mathbf{W}_Q^{(l)} = \begin{bmatrix} \mathbf{P}_t^{(l)} \mathbf{W}_Q^{(l)} \\ \mathbf{Q}_t^{(l)} \end{bmatrix}.$$

Similarly, we have the key states and value states

$$\hat{\mathbf{K}}_t^{(l)} = \begin{bmatrix} \mathbf{P}_t^{(l)} \mathbf{W}_K^{(l)} \\ \mathbf{K}_t^{(l)} \end{bmatrix}, \quad \hat{\mathbf{V}}_t^{(l)} = \begin{bmatrix} \mathbf{P}_t^{(l)} \mathbf{W}_V^{(l)} \\ \mathbf{V}_t^{(l)} \end{bmatrix}.$$

The attention output $\hat{\mathbf{A}}_t$ is then computed using a modified causal mask $\mathrm{Mask}(L_p + L) \in \mathbb{R}^{(L_p+L) \times (L_p+L)}$ by

$$\hat{\mathbf{A}}_t = \mathrm{Softmax}\left(\frac{\hat{\mathbf{Q}}_t^{(l)} \hat{\mathbf{K}}_t^{(l)\top}}{\sqrt{d_k}} + \mathrm{Mask}(L_p + L)\right) \hat{\mathbf{V}}_t^{(l)}.$$

Specifically, we divide the causal mask into four parts as follows

$$\text{Mask}(L_p + L) = \left[\begin{array}{c|c} \mathbf{0}_{L_p \times L_p} & -\infty_{L_p \times L} \\ \hline \mathbf{0}_{L \times L_p} & \text{Mask}(L) \end{array}\right],$$

where $\mathbf{0}_{L \times L_p}$ allows the virtual memory prefix tokens to be visible to input tokens. Then, we decompose the calculation of attention as

$$\hat{\mathbf{A}}_t = \begin{bmatrix} \text{Softmax}\left(\frac{\left(\mathbf{P}_t^{(l)}\mathbf{W}_Q^{(l)}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right) & \mathbf{0}_{L_p \times L} \\ \text{Softmax}^*\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right) & \text{Softmax}^*\left(\frac{\mathbf{Q}_t^{(l)}\mathbf{K}_t^{(l)\top}}{\sqrt{d_k}} + \text{Mask}(L)\right) \end{bmatrix} \begin{bmatrix} \mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)} \\ \mathbf{V}_t^{(l)} \end{bmatrix},$$

where $\text{Softmax}^*(\cdot)$ denotes the global softmax function applied to the entire row. Then, we retain the attention outputs corresponding to the non-virtual tokens by

$$\mathbf{A}_t^{(l)} = \underbrace{\text{Softmax}^*\left(\frac{\mathbf{Q}_t^{(l)}\mathbf{K}_t^{(l)\top}}{\sqrt{d_k}} + \text{Mask}(L)\right)\mathbf{V}_t^{(l)}}_{\text{Original Attention}} + \underbrace{\text{Softmax}^*\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right)}_{\text{Memory Attention}}. \tag{6}$$

Let $\mathbf{z}_{\text{orig}}$ and $\mathbf{z}_{\text{mem}}$ denote the partition items of the original attention and memory attention

$$\mathbf{z}_{\text{orig}} = \exp\left(\frac{\mathbf{Q}_t^{(l)}\mathbf{K}_t^{(l)\top}}{\sqrt{d_k}} + \text{Mask}(L)\right)\mathbf{1}_L, \quad \mathbf{z}_{\text{mem}} = \exp\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right)\mathbf{1}_{L_p}. \tag{7}$$

Then, we perform element-wise division by its element-wise sum to get the weighting matrices

$$\mathbf{\Lambda}_{\text{orig}} = \text{diag}\left(\mathbf{z}_{\text{orig}} \oslash \left(\mathbf{z}_{\text{orig}} + \mathbf{z}_{\text{mem}}\right)\right), \quad \mathbf{\Lambda}_{\text{mem}} = \text{diag}\left(\mathbf{z}_{\text{mem}} \oslash \left(\mathbf{z}_{\text{orig}} + \mathbf{z}_{\text{mem}}\right)\right) = \mathbf{I} - \mathbf{\Lambda}_{\text{orig}}.$$

Then, **Equation (6)** is equivalent to the equation with the normal Softmax function for each part:

$$\mathbf{A}_t^{(l)} = \mathbf{\Lambda}_{\text{orig}} \cdot \text{Softmax}\left(\frac{\mathbf{Q}_t^{(l)}\mathbf{K}_t^{(l)\top}}{\sqrt{d_k}} + \text{Mask}(L)\right)\mathbf{V}_t^{(l)} + \left(\mathbf{I} - \mathbf{\Lambda}_{\text{orig}}\right) \cdot \text{Softmax}\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right).$$

To control the influence of the two attention components, we introduce a global weighting parameter $\gamma \in [0, 1]$ to approximate the original weighting matrices by

$$\mathbf{A}_t^{(l)} = \gamma \cdot \text{Softmax}\left(\frac{\mathbf{Q}_t^{(l)}\mathbf{K}_t^{(l)\top}}{\sqrt{d_k}} + \text{Mask}(L)\right)\mathbf{V}_t^{(l)} + (1 - \gamma) \cdot \text{Softmax}\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right).$$

Then, we denote this specific memory attention part for $\mathbf{P}_t^{(l)}$ as

$$\check{\mathbf{A}}_t^{(l)} = \text{Softmax}\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right).$$

We define the function of similarity between $\mathbf{Q}_t^{(l)}$ and $\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}$ as

$$\text{Sim}\left(\mathbf{Q}_t^{(l)}, \mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right) = \exp\left(\frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\right).$$

Then, the memory attention can be rewritten as

$$\check{\mathbf{A}}_t^{(l)} = \operatorname{diag}\left(\operatorname{Sim}\left(\mathbf{Q}_t^{(l)}, \mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)\cdot \mathbf{1}\right)^{-1} \operatorname{Sim}\left(\mathbf{Q}_t^{(l)}, \mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right).$$

In order to decompose the $\mathbf{Q}_t^{(l)}$ part and $\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}$ part, we approximate the similarity with

$$\operatorname{Sim}\left(\mathbf{Q}_t^{(l)}, \mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right) = \frac{\mathbf{Q}_t^{(l)}\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}.$$

So the memory attention can be rewritten as

$$\check{\mathbf{A}}_t^{(l)} = \operatorname{diag}\left(\mathbf{Q}_t^{(l)}\frac{\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\cdot\mathbf{1}\right)^{-1}\mathbf{Q}_t^{(l)}\left[\frac{\left(\mathbf{P}_t^{(l)}\mathbf{W}_K^{(l)}\right)^\top}{\sqrt{d_k}}\left(\mathbf{P}_t^{(l)}\mathbf{W}_V^{(l)}\right)\right]. \tag{8}$$

Finally, we consider the prefix tokens $\mathbf{P}_t^{(l)}$ as the c-th ($c < t$) step aggregated results $\bar{\mathbf{H}}_c^{(l)}$, so we get

$$\check{\mathbf{A}}_t^{(l)} = \operatorname{diag}\left(\mathbf{Q}_t^{(l)}\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\cdot\mathbf{1}\right)^{-1}\mathbf{Q}_t^{(l)}\left[\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_V^{(l)}\right)\right], \tag{9}$$

where $\tilde{\mathbf{W}}_K^{(l)}, \tilde{\mathbf{W}}_V^{(l)}$ are parameters of the hyper memory block. It is worth noting that, in practice, the reference step c is not accessible in advance, and the evidence required at step t may span more than a single step. Meanwhile, the memory key states and memory value states of different steps are coupled together within the fixed-size memory network $\mathbf{M}_t^{(l)}$ and the normalization vector $\mathbf{S}_t^{(l)}$, so that the non-reference steps ($j \neq c$) inevitably leak into the readout as noise. To mitigate the impact of such noise, instead of directly reusing the vanilla attention query $\mathbf{Q}_t^{(l)}$ in **Equation (9)**, we utilize a trainable memory query $\tilde{\mathbf{Q}}_t^{(l)} = \tilde{\mathbf{H}}_t^{(l)}\tilde{\mathbf{W}}_Q^{(l)}$ to get

$$\check{\mathbf{A}}_t^{(l)} = \operatorname{diag}\left(\tilde{\mathbf{Q}}_t^{(l)}\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\cdot\mathbf{1}\right)^{-1}\tilde{\mathbf{Q}}_t^{(l)}\left[\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_V^{(l)}\right)\right], \tag{10}$$

where $\tilde{\mathbf{W}}_Q^{(l)} \in \mathbb{R}^{d\times d_k}$ is an optimizable projection. This decouples the memory query from the original attention and offers the freedom to reshape the cross-step similarities $\tilde{\mathbf{Q}}_t^{(l)}\tilde{\mathbf{K}}_j^{(l)\top}$. Therefore, it can emphasize the relevant steps while suppressing the irrelevant ones, thereby reducing the influence of noise when using the memory query states. We provide a detailed theoretical error analysis in **Section 3.5**.

3.5 Theoretical Error Analysis

Unlike standard Transformers that store all historical KV pairs in a growing cache, the hyper memory block compresses information into a fixed-size matrix. Suppose the model requires extracting information from the c-th step. The dense memory network can be expressed as

$$\mathbf{M}_t^{(l)} = \sum_{j=1}^{t-1}\lambda^{t-(j+1)}\cdot\frac{(1-\lambda)}{L'_j}\cdot\frac{\tilde{\mathbf{K}}_j^{(l)\top}}{\sqrt{d_k}}\tilde{\mathbf{V}}_j^{(l)}.$$

Then, we use the memory query $\tilde{\mathbf{Q}}_t^{(l)} = \tilde{\mathbf{H}}_t^{(l)}\tilde{\mathbf{W}}_Q^{(l)}$ to extract information from the dense memory network and query-key normalization vector by

$$\begin{aligned}\tilde{\mathbf{A}}_t^{(l)} =& \text{diag}\left(\tilde{\mathbf{Q}}_t^{(l)}\mathbf{S}_t^{(l)}\right)^{-1}\tilde{\mathbf{Q}}_t^{(l)}\mathbf{M}_t^{(l)} \\ =& \text{diag}\left(\sum_{j=1}^{t-1}\lambda^{t-(j+1)}\frac{(1-\lambda)}{L'_j}\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_j^{(l)\top}}{\sqrt{d_k}}\mathbf{1}\right)^{-1} \\ & \cdot\left(\sum_{j=1}^{t-1}\lambda^{t-(j+1)}\cdot\frac{(1-\lambda)}{L'_j}\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_j^{(l)\top}}{\sqrt{d_k}}\tilde{\mathbf{V}}_j^{(l)}\right).\end{aligned}$$

We define the individual terms in the above summation as

$$\mathbf{U}_j = \lambda^{t-(j+1)}\frac{(1-\lambda)}{L'_j}\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_j^{(l)\top}}{\sqrt{d_k}}\mathbf{1},$$

$$\mathbf{R}_{j'} = \lambda^{t-(j'+1)}\cdot\frac{(1-\lambda)}{L'_{j'}}\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_{j'}^{(l)\top}}{\sqrt{d_k}}\tilde{\mathbf{V}}_{j'}^{(l)},$$

and $\tilde{\mathbf{A}}_t^{(l)}$ can be rewritten as

$$\tilde{\mathbf{A}}_t^{(l)} = \text{diag}\left(\sum_{j=1}^{t-1}\mathbf{U}_j\right)^{-1}\left(\sum_{j'=1}^{t-1}\mathbf{R}_{j'}\right).$$

Assume that our target information is stored at the c-step (*i.e.,* the similarity $\tilde{\mathbf{Q}}_t^{(l)}\tilde{\mathbf{K}}_c^{(l)}$ is significantly higher than others). Then, we can further rewrite the equation as

$$\begin{aligned}\tilde{\mathbf{A}}_t^{(l)} =& \text{diag}\left(\mathbf{U}_c + \sum_{j=1,j\neq c}^{t-1}\mathbf{U}_j\right)^{-1}\left(\mathbf{R}_c + \sum_{j'=1,j'\neq c}^{t-1}\mathbf{R}_{j'}\right). \\ =& \left(\text{diag}\left(\mathbf{U}_c\right)\left(\mathbf{I} + \text{diag}\left(\mathbf{U}_c\right)^{-1}\cdot\text{diag}\left(\sum_{j=1,j\neq c}^{t-1}\mathbf{U}_j\right)\right)\right)^{-1}\left(\mathbf{R}_c + \sum_{j'=1,j'\neq c}^{t-1}\mathbf{R}_{j'}\right).\end{aligned}$$

According to the first-order Taylor expansion, we have

$$\begin{aligned}\tilde{\mathbf{A}}_t^{(l)} \approx& \left(\mathbf{I} - \text{diag}\left(\mathbf{U}_c\right)^{-1}\cdot\text{diag}\left(\sum_{j=1,j\neq c}^{t-1}\mathbf{U}_j\right)\right)\text{diag}\left(\mathbf{U}_c\right)^{-1}\left(\mathbf{R}_c + \sum_{j'=1,j'\neq c}^{t-1}\mathbf{R}_{j'}\right), \\ =& \text{diag}\left(\mathbf{U}_c\right)^{-1}\mathbf{R}_c + \underbrace{\text{diag}\left(\mathbf{U}_c\right)^{-1}\sum_{j'=1,j'\neq c}^{t-1}\mathbf{R}_{j'}}_{\boldsymbol{\epsilon}_1} \underbrace{- \text{diag}\left(\mathbf{U}_c\right)^{-1}\cdot\text{diag}\left(\sum_{j=1,j\neq c}^{t-1}\mathbf{U}_j\right)\text{diag}\left(\mathbf{U}_c\right)^{-1}\mathbf{R}_c}_{\boldsymbol{\epsilon}_2} \\ & \underbrace{- \text{diag}\left(\mathbf{U}_c\right)^{-1}\cdot\text{diag}\left(\sum_{j=1,j\neq c}^{t-1}\mathbf{U}_j\right)\text{diag}\left(\mathbf{U}_c\right)^{-1}\sum_{j'=1,j'\neq c}^{t-1}\mathbf{R}_{j'}}_{\boldsymbol{\epsilon}_3},\end{aligned}$$

where the first term is equivalent to $\check{\mathbf{A}}_t^{(l)}$ in **Equation (10)** with $\tilde{\mathbf{K}}_c^{(l)} = \bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}$ and $\tilde{\mathbf{V}}_c^{(l)} = \bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_V^{(l)}$ by

$$\begin{aligned}
\operatorname{diag}\left(\mathbf{U}_c\right)^{-1}\mathbf{R}_c =& \operatorname{diag}\left(\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_c^{(l)\top}}{\sqrt{d_k}}\mathbf{1}\right)^{-1}\tilde{\mathbf{Q}}_t^{(l)}\frac{\tilde{\mathbf{K}}_c^{(l)\top}}{\sqrt{d_k}}\tilde{\mathbf{V}}_c^{(l)} \\
=& \operatorname{diag}\left(\tilde{\mathbf{Q}}_t^{(l)}\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\mathbf{1}\right)^{-1}\tilde{\mathbf{Q}}_t^{(l)}\frac{\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_K^{(l)}\right)^\top}{\sqrt{d_k}}\left(\bar{\mathbf{H}}_c^{(l)}\tilde{\mathbf{W}}_V^{(l)}\right) \\
=& \check{\mathbf{A}}_t^{(l)}.
\end{aligned}$$

Therefore, there are three error terms for $||\tilde{\mathbf{A}}_t^{(l)} - \check{\mathbf{A}}_t^{(l)}||_2$. It is worth noting that ϵ_2 and ϵ_3 are structural errors caused by global normalization, whereas ϵ_1 is an attention error introduced by irrelevant information. Across these three terms, there is always at least one factor of $\tilde{\mathbf{Q}}_t^{(l)}\tilde{\mathbf{K}}_j^{(l)\top}$ where $j \neq c$ in the summation. Therefore, when this similarity is low, the resulting error is expected to be small.

3.6 Efficiency Analysis

Metis introduces native memory with limited additional inference overhead compared with external memory. The key reason is that the original attention, memory attention, and memory storage procedure can be largely executed in parallel. For the l-th layer at step t, the original attention branch computes token-token attention over the current input, while the memory utilization branch performs memory attention over the native memory state $\mathbf{M}_t^{(l)}$ and $\mathbf{S}_t^{(l)}$. These two branches depend on the same input hidden states but have no sequential dependency on each other. Therefore, memory attention does not need to wait for the output of the original attention, and its results can be fused only after both branches finish.

The memory storage procedure can also be decoupled from the current inference path. It updates the memory state for future steps, while the current step only reads from the existing memory state. Thus, after the required hidden states are available, the storage branch can be executed in parallel with the original attention and memory utilization, instead of becoming an additional sequential stage. As a result, the layer-level latency can be expressed as

$$T_{\text{parallel}}^{(l)} = \max\left(T_{\text{orig}}^{(l)}, T_{\text{util}}^{(l)}, T_{\text{store}}^{(l)}\right) + T_{\text{fuse}}^{(l)}. \tag{11}$$

Moreover, Metis stores historical information in fixed-size native memory states, rather than appending retrieved textual memories to the input context. Therefore, its memory utilization cost depends mainly on the memory state size, instead of growing linearly with the number of historical interactions. This enables Metis to provide native memory capabilities while avoiding the retrieval, concatenation, and prefilling overhead commonly introduced by external memory systems.

4 Data Construction

In order to build Metis by mid-training based on general foundation models, we synthesize a comprehensive training dataset based on existing public datasets. This dataset consists of primary data and auxiliary data, which are used for training native memory procedures and improving generalization in complex scenarios.

4.1 Primary Data

The primary data serves as the core supervision for training native memory procedures. It is designed to teach memory foundation models to perform different memory operations in the forward computation through optimization, thereby generalizing to various scenarios. The native memory procedure is acquired through optimization rather than manual rules, so the primary data must provide explicit supervision for the desired memory operations.

Data Principles. We highlight two data principles. First, the data should be structured as a temporally ordered sequence of interaction steps, which mirrors the time-streaming nature of online information. Second, the data should be state-consistent. The response to a later query must agree with the memory state shaped by earlier operations. Together, these two properties teach the model to store information and use it at the appropriate later step.

Table 1 Summary of the primary data. Samples for each memory operation are synthesized from public benchmarks and follow a distinct multi-turn skeleton, where the answer to the final query stays consistent with the preceding memory operations.

Operation	Interaction Streaming	Source Benchmarks
Remember	Info(A_1) → Query(A)	LoCoMo [27], LongMemEval [28], NeedleInAHaystack [29], RULER [30], LongBench [31], ∞Bench [32], L-Eval [33], BABILong [34], Bamboo [35], NaturalQuestions [36], LongChat-Eval [37]
Update	Info(A_1) → Info(A_2) → Query(A)	ZsRE [38], RippleEdits [39], KnowEdit [40], TemporalWiki [41]
Forget	Info(A_1) → Info($\bar{A}_1$) → Query(A)	TOFU [42], WMDP [43], MUSE [44], RWKU [45], WhoIsHarryPotter [46], BLUR [47], LKF [48], CLEAR [49], CounterFact [50]
Reflect	Info(A_1) → Info(B_1) → Query(A ∩ B)	MuSiQue [51], StrategyQA [52], Bamboogle [53]

Instead of generating data from scratch, we synthesize the primary data from established public benchmarks. This choice offers three advantages. Mature benchmarks provide verified facts and reasoning chains, which reduce hallucination when we extend them into long interaction sequences. Their broad coverage of fiction, science, news, and logical reasoning enriches the context and improves generalization. In addition, every synthetic sample is anchored to a source fact, which keeps the corpus traceable and easy to verify.

Data Summary. We select 27 public benchmarks across four memory operations, as shown in **Table 1**. We organize the primary data along three orthogonal dimensions: (1) Memory operation includes *remember*, *forget*, *update*, and *reflect*, which together span the core behaviors of native memory. For every operation, a structured fact serves as the unit of synthesis, and the final query is answerable only from the information introduced in the preceding turns. A *remember* sample states a fact and then queries it, whereas a *reflect* sample introduces several single-hop facts and then queries their multi-hop composition. An *update* sample modifies a previously stated fact before the query, and a *forget* sample revokes a previously stated fact before the query. (2) Salience of the instruction ranges from explicit memory commands to implicit statements that embed information within natural narratives. (3) Noise level, where clean sequences form the basic case and noisy sequences are produced by inserting irrelevant turns. Jointly, these dimensions encourage the memory procedure to generalize across operations, instruction styles, and noise levels.

Construction Pipeline. Our data synthesis pipeline comprises three major steps, including seed extraction, static synthesis, and quality verification.

Step 1: Seed Extraction. From each source dataset, we extract the source reference, query, and answer to form a base dialogue. Then, we summarize the underlying fact into a structured seed, which records a subject, a relation, and a target, together with operation-specific fields such as the updated target or the multi-hop chain. We also collect a pool of distractor dialogues that are logically orthogonal to each query, which are used to extend the sequence length.

Step 2: Static Synthesis. Guided by the structured seed, a strong instruction-following language model rewrites each base dialogue into two salience styles. The explicit style phrases the reference as a clear memory instruction, while the implicit style states the same fact as a description without an explicit instruction. To cover long-range memory, we insert a variable number of distractor turns between the reference and the query, which yields the distract variant of both styles.

Step 3: Quality Verification. A language model acts as an automatic judge and filters samples according to several quality criteria. The consistency check confirms that the final answer faithfully reflects the intended memory state. The orthogonality check ensures that inserted distractors do not leak the core fact, and the shortcut check removes any query that can be answered without its reference. We additionally monitor the semantic diversity of the queries to prevent template collapse. Samples that fail any check are discarded, so that only reliable samples enter the final corpus.

Data Statistics. We report the statistics of the synthesized primary data in **Table 2**. After filtering, the corpus contains 357,137 samples and about 406 million tokens, drawn from 27 source benchmarks. The samples are distributed across explicit, implicit, and distractor styles, which balance instruction salience and noise level. The token count is dominated by the distractor samples, especially for *remember*, because long irrelevant contexts are inserted to strengthen long-range memory. This profile indicates that the primary data covers diverse memory operations at varied interaction lengths, which provides a solid basis for training native memory procedures.

Table 2 Statistics of the synthesized primary data. The explicit, implicit, and distract columns report the number of samples of each style, and the last column reports the total token count in millions.

Operation	Sources	Explicit	Implicit	Distract	All Samples	Tokens (M)
Remember	11	14,682	13,671	28,502	56,855	362.0
Forget	9	59,900	8,251	68,120	136,271	21.7
Update	4	33,452	7,300	40,749	81,501	11.0
Reflect	3	20,646	20,615	41,249	82,510	11.4
Total	**27**	**128,680**	**49,837**	**178,620**	**357,137**	**406.1**

Table 3 Summary of the auxiliary data. Each subtype composes facts or dialogues into a complex interaction pattern, where the final answers remain consistent with the intended memory state.

Auxiliary Subtype	Interaction Streaming	Construction Source
Multi-Entity Binding	Info(A_1) → Info(B_1) → Query(A) → Query(B)	Paired facts synthesized from primary source facts
Selective Forgetting	Info(A_1) → Info(B_1) → Info($\bar{B}_1$) → Query(B) → Query(A)	Paired facts synthesized from primary source facts
Post-Memory Dialogue	Info(A_1) → Query(A) → Chat	Primary memory samples with curated normal dialogues
Memory-Irrelevant Dialogue	Info(A_1) → Chat / Chat → Chat	Primary memory samples with curated normal dialogues

4.2 Auxiliary Data

The auxiliary data is used to improve the model's generalizability. It further enhances the capabilities of memory foundation models for complex scenarios, such as multi-entity tasks and mixed dialogues.

Construction Principles. The primary data consists of basic memory operations interactions, ranging from single-fact operations to multi-fact reasoning cases. However, real interactions are more complex. Multiple similar facts may coexist, some facts may be revoked while others persist, and memory turns are often interleaved with ordinary conversation.

The first is interference, where the model confuses similar facts or allows a forgetting operation to corrupt a retained fact in parametric spaces. The second is memory pollution, where the model applies stored values to questions that do not need them. The auxiliary data complements the primary data by targeting exactly these scenarios. It preserves the same fact-level structure, but composes facts and dialogues into more challenging interaction patterns, which improves the generalization and robustness of native memory.

Data Summary. We organize the auxiliary data into four subtypes, shown in **Table 3**. The first two subtypes address multi-fact scenarios. (1) Multi-entity binding jointly stores two confusable facts and queries both, which trains the model to bind each value to its own fact. (2) Selective forgetting revokes one fact while the other persists, which trains the model to forget one fact selectively without collateral loss. The other two subtypes address memory pollution. (3) Post-memory dialogue continues an ordinary conversation right after a memory query, so the model does not carry stored values into unrelated answers. (4) Memory-irrelevant dialogue answers a question that does not need memory even when a memory state exists, so the model learns when memory should not influence the response. Therefore, these four subtypes extend the primary data to realistic mixed interactions.

Construction Pipeline. The auxiliary data is built from two shared ingredients, including synthesized paired facts and prepared normal dialogues, which are then formulated into the four subtypes.

Paired Fact Synthesis. The multi-entity binding and selective forgetting subtypes require pairs of similar facts. For each source fact, represented by a subject, relation, and value, we synthesize one confusable counterpart fact. Each counterpart is generated using one of four transformations relative to the source fact. It keeps the subject but changes the relation, keeps the relation but changes the subject, imitates the value format, or stays semantically adjacent. A language model generates each counterpart, and a verifier discards any fact that contradicts, restates, or depends on the source. We then rewrite the verified facts into natural statements, queries, and revocation snippets to ensure fluency and diversity.

Table 4 Statistics of the synthesized auxiliary data. We report the number of samples for each subtype, and we exclude every sample that fails quality verification.

Target	Auxiliary Subtype	Samples
Multi-fact Scenario	Multi-Entity Binding	76,153
	Selective Forgetting	76,153
Memory Pollution	Post-Memory Dialogue	357,137
	Memory-Irrelevant Dialogue	100,000
All	**Total**	**609,443**

Normal Dialogue Preparation. The post-memory dialogue and memory-irrelevant dialogue subtypes require conversations that do not require access to memory. We prepare a dialogue pool from three sources. These are general assistant dialogues for everyday requests, open-domain conversations from public corpora, and entity-related dialogues that are topically related to a stored fact yet remain answerable without it. We filter out turns with memory cues, real-time facts, or unsafe content, and we remove duplicates.

Subtype Formulation. The multi-entity binding subtype states the paired facts in turn and then queries both, which forces the model to bind each value to its correct fact. The selective forgetting subtype states both facts, revokes one, and then queries both, so the revoked fact becomes unavailable while the retained fact stays correct. The post-memory dialogue subtype appends an unrelated ordinary turn after a memory query, so the model returns to normal conversation without leaking any stored value. The memory-irrelevant dialogue subtype keeps the memory state but drops its query before an ordinary turn, and it also includes standalone dialogues that carry no memory at all.

Data Statistics. We report the statistics of the synthesized auxiliary data in **Table 4**. The paired-fact synthesis yields 76,153 natural snippet sets after quality verification. These snippets support 76,153 multi-entity binding samples and 76,153 selective forgetting samples. The dialogue-based subtypes are larger, because they reuse the full set of primary memory samples. Post-memory dialogue contributes 357,137 samples, and memory-irrelevant dialogue contributes 100,000 samples. In total, the auxiliary data adds 609,443 samples that emphasize multi-fact reasoning and pollution-resistant conversation. Together with the primary data, it provides broad coverage from single-fact operations to complex mixed interactions.

5 Model Optimization

To empower Metis with native memory procedures, we design multiple training objectives for mid-training. These objectives primarily consist of memory reconstruction, memory operation, and regularization. The three objectives share a common likelihood form but operate on different data. They jointly shape the native memory state and procedure.

5.1 Overview

We organize every training sample as a multi-step interaction $s = \{(X_t, Y_t)\}_{t=1}^{T_s}$, following the definition in **Section 2**. At step t, the model reads the input instruction X_t and generates the assistant response $Y_t = (y_{t,1}, \ldots, y_{t,|Y_t|})$. All steps are forwarded sequentially, and the native memory procedure stores information from each step to the memory state before the next step starts. Therefore, the parameters θ_t at step t already integrate the memory state from all preceding steps $\{(X_i, Y_i)\}_{i<t}$. We supervise only a subset of query steps $Q_s \subseteq \{1, \ldots, T_s\}$, where the assistant response is labeled, while the reference and operation steps remain unlabeled. However, the responses of the reference and operation steps are still generated or provided for memory state updates. In addition, all three objectives share the per-step loss below, and they differ only in how the supervised target Y_t is constructed from the training data. For a supervised step $t \in Q_s$, we define the per-step loss as the token-averaged negative log-likelihood

$$\ell(s,t) = -\frac{1}{|Y_t|} \sum_{k=1}^{|Y_t|} \log P(y_{t,k} \mid X_t, Y_{t,<k}; \theta_t), \tag{12}$$

where $Y_{t,<k}$ denotes the previously generated tokens at step t, and θ_t is conditioned on the memory state shaped by earlier steps. The loss of a sample aggregates over its supervised steps as $\sum_{t \in Q_s} \ell(s,t)$, which allows a single trajectory

to supervise multiple responses. During mid-training, we freeze the backbone parameters and optimize only the native memory parameters.

The three objectives correspond to five data subsets, and we control their contributions through a task-weighted sampler rather than explicit loss coefficients. At training epoch e, the sampling probability of subset τ is

$$\pi_\tau(e) = \frac{w_\tau(e)}{\sum_{\tau' \in \mathcal{T}} w_{\tau'}(e)}, \qquad w_\tau(e) = w_\tau^{\mathrm{s}} + \left(w_\tau^{\mathrm{e}} - w_\tau^{\mathrm{s}}\right) \cdot \min\left(\frac{e}{E-1}, 1\right), \tag{13}$$

where $\mathcal{T}$ is the set of subsets, E is the total number of epochs, and $w_\tau^{\mathrm{s}}, w_\tau^{\mathrm{e}}$ are the start and end weights of subset τ. This linear annealing forms a curriculum that gradually shifts the sampling mass from storage-oriented data toward harder long-range and regularization data. Because the weights only modulate the sampling frequency, the expected mid-training objective can be written as

$$\mathcal{L} = \sum_{\tau \in \mathcal{T}} \pi_\tau(e) \cdot \mathbb{E}_{s \sim \mathcal{D}_\tau}\left[\sum_{t \in Q_s} \ell(s,t)\right], \tag{14}$$

where $\mathcal{D}_\tau$ is the data of subset τ, and every sampled step contributes an unweighted loss from **Equation (12)**.

5.2 Memory Reconstruction Objective

The memory reconstruction objective enables Metis to store and reconstruct information. It provides an important training signal during the model's warm-up phase, as initialized models typically lack such capabilities. Furthermore, it targets the upper bound of information storage, with completely lossless compression and reconstruction. However, a trade-off exists between this objective and the native memory procedure. First, reconstruction and instruction following are contradictory, as they require specificity and generalization, respectively. Second, from the perspective of prediction tasks, the native memory procedure requires lossy compression guided by input instructions. In contrast, memory reconstruction opposes lossy compression.

This objective is built on a reconstruction subset derived from the primary data in **Section 4.1**, denoted as $\mathcal{D}_{\mathrm{rec}}$. In each sample, a reference passage is presented and stored into the memory state at an early step, and a later query step requires the model to regenerate its content. Because the supervised response Y_t reproduces the stored reference, the memory state must retain the source with minimal loss. We instantiate the per-step loss over this subset as

$$\mathcal{L}_{\mathrm{rec}} = \pi_{\mathrm{rec}}(e)\mathbb{E}_{s \sim \mathcal{D}_{\mathrm{rec}}}\left[\sum_{t \in Q_s} \ell(s,t)\right], \tag{15}$$

where the expectation averages over samples drawn from $\mathcal{D}_{\mathrm{rec}}$, and $\ell(s,t)$ measures the negative log-likelihood of reconstructing the stored content at the query step t. By step t, the reference passage has already been stored in the native memory state represented within θ_t. Minimizing $\mathcal{L}_{\mathrm{rec}}$ thus drives the hyper memory block to encode the reference into a state from which the memory utilization procedure can recover it.

5.3 Memory Operation Objective

While reconstruction establishes lossless storage, native memory must additionally support input-driven operations. The memory operation objective teaches Metis to remember, forget, update, and reflect, so that the memory state evolves according to the instruction at each step. It is built on the primary data, which exhibits these operations under controlled instruction salience and noise.

We use two complementary subsets of the primary data. The first subset, denoted as $\mathcal{D}_{\mathrm{op}}^{\mathrm{e/i}}$, contains the explicit and implicit samples. Explicit samples phrase the operation as a clear command, whereas implicit samples embed the same information within a natural narrative. This contrast forces the model to infer the operation from intent rather than from surface keywords. The second subset, denoted as $\mathcal{D}_{\mathrm{op}}^{\mathrm{d}}$, contains the distractor samples, where irrelevant turns are inserted between the reference and the query. It promotes long-range retention and robustness against intervening noise.

In all operation samples, the supervised response stays consistent with the information from the earlier steps. For an *update* sample, the answer reflects the new value rather than the old one. For a *forget* sample, the answer no longer

exposes the forgotten value. For a *reflect* sample, the answer composes several stored facts into multi-hop reasoning. Therefore, a single likelihood objective suffices to supervise all operations as

$$\mathcal{L}_{\text{op}} = \pi_{\text{e/i}}(e)\mathbb{E}_{s\sim\mathcal{D}_{\text{op}}^{\text{e/i}}}\left[\sum_{t\in Q_s}\ell(s,t)\right] + \pi_{\text{d}}(e)\mathbb{E}_{s\sim\mathcal{D}_{\text{op}}^{\text{d}}}\left[\sum_{t\in Q_s}\ell(s,t)\right], \tag{16}$$

where θ_t now encodes the net effect of the preceding operation sequence on the memory state. Unlike reconstruction, the target is no longer a copy of the stored content, so the model learns to transform and read the memory state under the guidance of the instruction.

5.4 Regularization Objective

The reconstruction and operation objectives are primarily built on simple interaction patterns, which leave the model vulnerable in complex scenarios. The regularization objective mitigates two failure modes that arise when memory operates in realistic interactions. The first is interference, where similar facts are confused or a forgetting operation corrupts a retained fact. The second is memory pollution, where stored values leak into responses that do not require them [54]. This objective is built on the auxiliary data in **Section 4.2**, which composes facts and dialogues into more complex and realistic interaction patterns.

We use two subsets of the auxiliary data. The multi-fact subset $\mathcal{D}_{\text{mf}}$ targets interference. Its multi-entity binding samples jointly present two confusable facts and query the model about both, which constrains the memory utilization procedure to bind each value to its own key. Its selective forgetting samples include an instruction that revokes one fact while preserving the other, which constrains the forget operation to act locally. The memory pollution subset $\mathcal{D}_{\text{mp}}$ targets leakage. Its post-memory dialogue samples continue an ordinary conversation right after a memory query, and its memory-irrelevant samples answer a question that needs no memory even when a memory state exists. In both cases, they discourage the model from injecting memory into unrelated responses.

These subsets act as regularization because they constrain memory behavior under more realistic and diverse interaction scenarios. The supervised targets penalize cross-fact interference, collateral forgetting, and value leakage, which suppress degenerate solutions that always read or overwrite the memory state. We define the objective as

$$\mathcal{L}_{\text{reg}} = \pi_{\text{mf}}(e)\mathbb{E}_{s\sim\mathcal{D}_{\text{mf}}}\left[\sum_{t\in Q_s}\ell(s,t)\right] + \pi_{\text{mp}}(e)\mathbb{E}_{s\sim\mathcal{D}_{\text{mp}}}\left[\sum_{t\in Q_s}\ell(s,t)\right], \tag{17}$$

where many samples expose multiple supervised steps, so $|Q_s| > 1$ jointly constrains the retained and the revoked facts within one interaction. For the memory-irrelevant case, the supervised step is an ordinary turn whose target is independent of the memory state.

6 Experiments

6.1 Experimental Settings

We evaluate Metis on memory operation tasks and memory-based question-answering (QA) tasks. The memory operation task evaluates the performance of executing memory operations. In addition, to verify the effectiveness of the native memory state, we evaluate the performance on the memory-based QA task. Our major experiments focus on evaluating the native memory state and procedure primarily through relatively short-term tasks. As for the long-term capability, we explore it from the perspective of memory capability in **Section 6.6**.

Datasets and Metrics. For memory operations, we employ MemOps [55], which is a specific benchmark focusing on memory operations, such as remembering, forgetting, and updating. In the Full setting, the model receives three complete evidence segments, containing 24 utterances. In the Gold setting, it receives only the oracle turns required for the question. We also present the performance of the Test set of our constructed dataset. For the memory-based QA task, we conduct experiments on the golden-session setting of LoCoMo (*i.e.,* LoCoMo (Gold)), where we provide the gold evidence sessions as input. We also utilize the contextual generation task dataset from NextMem [56] for further analysis. This dataset evaluates whether models can utilize the provided information to answer questions correctly, consisting of SQuAD [57], HotpotQA [58], LoCoMo [27], and LongMemEval [28]. In all these settings, we utilize

Table 5 The overall performance on memory operation tasks. Full-context and partial-context results are shown in gray to visually distinguish context-access settings from the no-context comparison. Within the No Context setting, the best and second-best scores are **bolded** and underlined, respectively. Avg. represents the micro-average performance.

Type	Method	MemOps (Gold)					Metis Test Set				
		Remember	Update	Forget	Reflect	Avg.	Remember	Update	Forget	Reflect	Avg.
Full Context	Qwen3.5-4B	84.97	86.34	81.36	85.17	84.56	80.07	70.31	70.42	83.13	75.18
	Qwen3.5-9B	88.54	88.43	82.73	86.90	86.86	78.18	69.48	67.50	84.38	73.89
	Qwen3.5-27B	91.37	90.74	84.32	84.48	87.90	81.01	73.44	75.83	88.75	78.87
Partial Context	Qwen3.5-4B	38.84	33.56	24.55	21.90	30.18	70.05	63.12	59.90	67.81	64.82
	Qwen3.5-9B	30.51	26.62	20.23	11.55	22.41	70.40	55.10	55.00	64.69	60.68
	Qwen3.5-27B	37.05	35.88	22.05	16.21	28.01	70.28	63.33	66.46	60.47	65.40
No Context	Qwen3.5-4B	4.17	0.00	1.59	0.00	1.65	12.03	0.00	49.58	0.00	16.96
	Qwen3.5-9B	4.17	1.85	0.91	0.00	1.88	11.79	5.42	49.90	0.63	18.64
	Qwen3.5-27B	3.57	0.93	0.91	0.69	1.69	10.73	2.08	47.50	1.25	16.87
	Temp-LoRA-4B	15.33	10.19	2.95	4.83	8.85	15.80	27.71	15.21	17.66	19.34
	Temp-LoRA-9B	23.81	13.43	5.00	8.10	13.51	20.05	17.71	20.21	20.47	19.51
	Temp-LoRA-27B	20.68	6.48	2.50	4.83	9.70	25.94	18.65	25.21	26.87	23.86
	δ-Mem	7.44	6.02	1.82	1.55	4.38	13.92	21.77	12.40	10.31	15.03
	Metis-4B	19.35	27.55	7.27	16.90	17.84	52.24	63.85	31.25	90.16	56.72
	Metis-9B	25.89	23.61	**11.59**	15.52	19.63	58.14	63.33	30.42	90.78	57.92
	Metis-27B	**28.27**	**31.02**	10.91	**26.55**	**24.76**	**61.08**	**68.13**	**77.50**	**93.44**	**73.77**

gpt-4.1-mini to judge each prediction against its reference answer in three repeated evaluations. Then, we report the median LLM-as-a-judge score. In each dataset, we calculate the average performance (*i.e.,* Avg.) across different types using a micro-average. It should be noted that, to cover a wide range of entities for memory, we extract seed entities from various public datasets to synthesize our training data, such as LoCoMo and LongMemEval. However, we do not leak their exact QA behaviors in the training phase.

Baselines. We comprehensively evaluate our approach against four categories of baselines. For backbone models evaluated with full information appended to the context, we utilize Qwen3.5 [59] across 4B, 9B, and 27B sizes. For the partial-context baselines, we apply RAG [60] to these backbone models. It encodes observations and queries into dense representations, and calculates the cosine similarity between queries and all observations. The top-5 observations are appended to the context. For TTT-based models, we evaluate Temp-LoRA [61] as the baseline. It fuses information into the model by training a temporary LoRA module on previous text chunks during inference, encoding historical context as transient parameter updates. Specifically, we implement Temp-LoRA with corresponding sizes of Qwen3.5 backbones. Regarding parametric memory models, we compare with δ-Mem [22], which steers attention with low-rank corrections. More details are provided in Appendix E.

Training Configuration. The reported Metis models are built upon Qwen3.5 backbones and trained on 8×H100 GPUs. The backbone is frozen during training, and the trainable memory parameters are initialized using the key and value projection matrices of the corresponding backbone layers. We use AdamW with a learning rate of 2×10^{-4}, a constant schedule after 200 warmup steps, weight decay 0.01, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and gradient clipping at 1.0. Training uses BF16 and seed 42, and saves a checkpoint every 2,000 steps. For Metis-4B, we train our model for 14,000 steps, corresponding to one epoch. For Metis-27B, we use the same number of training steps, corresponding to approximately 0.4 epochs. For Metis-9B, we use 8,000 steps (approximately 0.5728 epochs), which is selected by early stopping on validation-set performance.

Evaluation Pipeline. For the memory operation and memory-based QA tasks, we adopt a static evaluation paradigm. Each test trajectory is divided into two sequential phases. The first phase consists of information steps, which provide the necessary context to the model. The second phase consists of query steps, where the model must answer a question based on the prior information. Finally, the evaluation calculates performance metrics by comparing the model's output in the query step with the ground truth. The prompts of the information step and query step are provided in **Appendix F.2**, and the prompts of LLM-as-a-Judge are presented in **Appendix F.3**.

Table 6 The overall performance on memory-based QA tasks. Full-context and partial-context results are shown in gray to visually distinguish context-access settings from the no-context comparison. Within the No Context setting, the best and second-best scores are **bolded** and underlined, respectively. Single and Multi indicate Single-hop Retrieval and Multi-hop Retrieval settings in LoCoMo (Gold), respectively. Temporal and Open refer to the temporal reasoning setting and open domain knowledge settings in LoCoMo (Gold), respectively. A dash indicates that the corresponding result is not applicable. Avg. represents the micro-average performance.

Type	Method	LoCoMo (Gold)					NextMem				
		Single	Multi	Temporal	Open	Avg.	SQuAD	HotpotQA	LongMemEval	LoCoMo	Avg.
Full Context	Qwen3.5-4B	85.12	65.92	15.78	23.88	63.52	91.00	88.28	45.21	61.00	77.15
	Qwen3.5-9B	84.43	64.93	16.64	21.35	63.00	91.00	87.61	45.79	60.65	77.05
	Qwen3.5-27B	85.83	69.96	15.55	31.18	65.03	91.80	89.18	48.43	64.47	78.80
Partial Context	Qwen3.5-4B	36.28	10.79	6.17	5.62	23.54	-	-	-	-	-
	Qwen3.5-9B	34.05	8.27	7.50	2.81	21.97	-	-	-	-	-
	Qwen3.5-27B	35.63	10.07	7.73	1.97	23.17	-	-	-	-	-
No Context	Qwen3.5-4B	0.00	0.36	0.00	1.97	0.18	11.24	25.22	2.86	0.48	11.86
	Qwen3.5-9B	0.00	0.36	0.00	0.00	0.07	14.92	34.98	2.86	0.48	15.93
	Qwen3.5-27B	0.00	0.36	0.00	0.00	0.07	16.73	38.90	3.14	0.48	17.75
	Temp-LoRA-4B	10.92	11.24	1.80	26.69	9.99	26.19	38.62	9.71	11.12	24.20
	Temp-LoRA-9B	13.33	13.31	2.42	25.00	11.72	29.52	45.07	12.93	12.80	28.12
	Temp-LoRA-27B	4.29	5.49	1.25	10.67	4.24	37.68	51.46	6.57	5.86	30.97
	δ-Mem	12.86	10.16	3.28	20.22	10.79	20.74	33.02	9.79	10.29	20.42
	Metis-4B	18.90	15.29	7.03	28.37	16.31	29.62	58.13	39.36	50.48	41.69
	Metis-9B	18.87	18.53	7.03	27.25	16.81	33.06	63.45	33.36	51.56	43.39
	Metis-27B	**31.01**	**27.97**	**13.83**	**28.93**	**26.74**	**43.42**	**66.54**	**39.71**	**60.41**	**50.82**

6.2 Overall Performance

Memory Operation Tasks. The results of MemOps in the gold setting (*i.e.,* MemOps (Gold)) and the Metis test set are presented in **Table 5**. Due to the page limitation, we put the experiment results and analysis of MemOps in the full setting (*i.e.,* MemOps (Full)) in **Appendix B**. As expected, full-context models achieve the strongest overall performance, while removing the context causes a substantial performance drop for standard backbones. Partial context preserves some information on the Metis test set but performs poorly on MemOps (Gold), showing that incomplete histories cannot reliably support memory operations. Temp-LoRA and δ-Mem recover part of the lost performance, but their gains remain limited. Under the same no-context setting, Metis achieves the best average results on both MemOps (Gold) and the Metis test set. These results suggest that Metis can preserve information in its native memory state and use it in later steps without replaying the original context.

Metis-27B achieves the best average performance on both benchmarks under the no-context setting. Compared with Metis-4B and Metis-9B, it shows clear gains in remembering, updating, reflection, and overall performance. The improvement is especially large for forgetting on the Metis test set. These results suggest that a sufficiently large backbone can better formulate and utilize the native memory state. In addition, forgetting is still the most difficult operation on the external MemOps (Gold) benchmark, even for Metis-27B. This suggests that removing or suppressing information in a shared latent state is more difficult to generalize than storing or updating information. Overall, Metis shows strong performance on memory operation tasks in short-term scenarios.

Memory-based QA Tasks. The results of the memory-based QA tasks are presented in **Table 6**. Full-context models provide a strong upper bound because they can directly attend to the original evidence. Their performance drops sharply when only partial context is available. Without context, the original Qwen3.5 models obtain almost zero scores on LoCoMo (Gold), confirming that the answers cannot be reliably recovered from backbone knowledge alone. In contrast, Metis achieves the best average performance on both benchmarks under the no-context setting. Metis-27B obtains the highest score in every task category, outperforming other baselines. These results show that the native memory state can preserve useful information and support question answering without replaying the original context.

The advantage of Metis is especially clear on tasks with complex or long-range memory requirements. On NextMem, Metis achieves large gains on HotpotQA, LongMemEval, and LoCoMo subset. It also substantially improves multi-hop and temporal question answering on LoCoMo (Gold). It indicates that native memory remains effective for relatively simple factual questions while providing larger gains on more demanding tasks. The strong improvement on temporal questions also suggests that a larger Metis model can better preserve and use relations across different interaction steps.

However, the gain on open-domain LoCoMo (Gold) questions is relatively limited, which indicates that some task types remain difficult even with increased model capacity.

In addition, we find that the improvement between Metis-4B and Metis-9B is modest, whereas Metis-27B substantially improves the average score. This pattern suggests that backbone scaling can enhance native memory capability once model capacity is reached, although the gains are not uniform across tasks. These results indicate that Metis provides strong memory-based QA performance in both the short-term QA tasks and the relatively longer LoCoMo (Gold) setting. In **Appendix D**, we further apply Metis to Llama [62] and Gemma [63] models of varying sizes. We use the same mid-training and evaluation paradigm to explore its transferability across different backbone families and scales.

6.3 Ablation Studies

We conduct ablation studies on Metis-4B from the perspectives of training data and model structure. Following the main experimental setup, we evaluate LoCoMo (Gold) and NextMem in memory-based QA tasks. We also use the Metis test set and MemOps (Gold) in memory operation tasks. All ablation models use Metis-4B and the same training configuration and evaluation pipeline as the main results.

Data Ablation. We evaluate the contribution of different training data through two variants. In w/o MS, we remove the Multi-fact Scenario data. In w/o MS+MP, we remove the entire auxiliary dataset, including Multi-fact and Memory Pollution data, to examine its overall contribution to memory learning and generalization.

As shown in **Table 7**, removing the Multi-fact Scenario data consistently reduces performance across both types of tasks. This result indicates that multi-fact supervision helps Metis integrate related information and maintain a coherent memory state. The decline is more evident on MemOps (Gold) and the Metis test set, suggesting that such data is particularly important for learning reliable memory operations. Removing the entire auxiliary dataset leads to a much larger overall degradation. The drop is especially clear on the Metis test set, while performance on LoCoMo (Gold) and NextMem also decreases consistently. This shows that auxiliary data improves the robustness and generalization of native memory procedures across different scenarios.

Structure Ablation. We further ablate the main components of the native memory procedure. In w/o GDU, we replace the GDU with a linear update (LU). In w/o SA, we remove the adaptive aggregation mechanism and directly use the last-token hidden state for memory storage. In w/o OQ, we remove the optimizable memory query projection and reuse the query from the original attention. In w/o QKN, we remove query-key normalization from memory attention.

Among all these evaluated variants, removing adaptive aggregation causes the largest performance drop. Directly using the last token cannot effectively capture information distributed across the input sequence. As a result, the model fails to construct an informative memory state. In addition, removing query-key normalization also causes a substantial degradation, particularly on LoCoMo (Gold) and NextMem. Without this normalization, irrelevant information may introduce stronger interference. Furthermore, reusing the original attention query also reduces performance across all benchmarks. The decrease is larger on the memory-based QA tasks, indicating that a separate memory query is important for distinguishing relevant historical information from noise. This observation is consistent with our theoretical analysis, where the additional query projection reshapes cross-step similarities and suppresses interference from irrelevant memory.

We also find that replacing the GDU with a linear update has a small effect on the overall average. The linear update performs slightly better on MemOps (Gold) and the Metis test set but is clearly weaker on LoCoMo (Gold). This suggests that a linear update can handle simple and short-term memory operations, while the GDU provides a better balance in long-term scenarios. This result is consistent with our engineering observation that GDU may produce more stable model behavior, motivating its use in Metis. **Appendix C.1** further compares LU and GDU across model scales, while **Appendix C.2** repeats the data and structure ablations from the LU baseline.

6.4 Out-of-Distribution Memory Tasks

To examine whether the strong performance reported in **Section 6.2** generalizes beyond the data-construction distribution, we further evaluate Metis on two out-of-distribution (OOD) benchmarks that were not used to construct the training data. All methods are evaluated under the no-context setting.

Table 7 Results of the ablation studies on Metis-4B. We use Avg. to report the macro-average performance in different groups, and utilize ΔAvg. to represent the relative performance gap compared with the full model.

Type	Model	Memory Operation Task				Memory-based QA Task				Overall	
		MemOps (Gold)	Metis Test Set	Avg.	ΔAvg.	LoCoMo (Gold)	NextMem	Avg.	ΔAvg.	Avg.	ΔAvg.
Full Model	Metis	17.84	56.72	37.28	-	16.31	41.69	29.00	-	33.14	-
Data Ablation	w/o MS	14.64	51.53	33.08	-11.26%	14.78	37.79	26.29	-9.36%	29.68	-10.43%
	w/o MS+MP	14.45	42.17	28.31	-24.07%	14.18	36.17	25.17	-13.20%	26.74	-19.31%
Structure Ablation	w/o GDU	18.50	58.54	38.52	3.32%	11.97	42.78	27.37	-5.60%	32.95	-0.58%
	w/o SA	3.67	19.72	11.70	-68.63%	9.84	18.49	14.16	-51.16%	12.93	-60.98%
	w/o OQ	13.89	53.93	33.91	-9.04%	11.46	37.07	24.26	-16.33%	29.09	-12.23%
	w/o QKN	9.32	48.74	29.03	-22.13%	10.00	26.80	18.40	-36.55%	23.72	-28.44%

Table 8 Results on OOD memory benchmarks. The best and unique second-best scores are **bolded** and underlined, respectively. The average score is calculated according to the official category counts.

Method	ATM-Bench (Gold)				MemDaily (Gold)						
	List	Number	Open	Avg.	Aggreg.	Comp.	Cond.	Noisy	Post-proc.	Simple	Avg.
δ-Mem	0.00	1.94	3.11	2.27	29.44	21.14	44.40	38.40	59.00	45.58	39.84
Temp-LoRA-4B	0.00	0.00	5.06	2.57	30.74	30.49	50.80	43.60	60.60	52.01	44.92
Temp-LoRA-9B	0.00	0.00	5.64	2.86	45.24	31.91	58.40	46.40	66.80	59.04	51.42
Temp-LoRA-27B	0.00	0.00	5.06	2.57	**61.90**	40.24	**61.00**	50.60	74.20	**68.67**	**59.45**
Metis-4B	**1.08**	14.17	9.92	10.22	45.02	54.67	51.00	44.60	64.80	54.62	52.54
Metis-9B	0.00	24.72	**15.18**	16.49	30.30	34.35	53.00	43.60	66.80	54.22	47.29
Metis-27B	0.00	**31.39**	14.59	**18.56**	40.69	**66.06**	56.60	**52.80**	**75.40**	61.45	59.04

We evaluate ATM-Bench [64] on its official standard split. For MemDaily [65], we use the subset of the official pre-generated release in which the annotated retrieval-target messages occur before the query. In the Gold setting, the model receives only benchmark-annotated evidence: human-annotated memory items represented as text in SGM for ATM-Bench, and retrieval-target messages for MemDaily. ATM-Bench scores list-recall, number, and open-ended questions using Jaccard similarity, post-processed exact match, and an LLM judge, respectively. MemDaily reports deterministic single-choice accuracy for all six question types.

As shown in **Table 8**, Metis demonstrates strong OOD transfer on ATM-Bench, consistently outperforming the memory baselines across model scales and most question types. The advantage also holds for the deterministically scored number questions, indicating that the improvement is not merely an artifact of the LLM judge used for open-ended questions. Since ATM-Bench requires models to retain and integrate heterogeneous evidence extracted from personal archives, these results suggest that the native memory procedure learned by Metis transfers beyond the patterns observed during training. The results on MemDaily are more mixed: Metis remains competitive but does not consistently lead the memory baselines. Together, the two benchmarks provide evidence that Metis's native memory capability generalizes to benchmarks not used in constructing its training data.

6.5 Source-Exclusion Study

Complementing the OOD evaluation, we study how sensitive Metis is to the composition of the public sources used by the synthesis pipeline. Specifically, we remove all training instances generated from LoCoMo and LongMemEval, and train Metis-4B, Metis-9B, and Metis-27B on the remaining data. We keep other training and evaluation configurations unchanged. The evaluation is memory-only, without replaying the original context. Under the accounting used for this experiment, the exclusion removes only about 2.61% of training instances, but these removals are concentrated in the remember, reconstruction, and multi-entity or mixed-operation slices.

As shown in **Table 9**, the six-benchmark macro-average decreases at all three model sizes, but the reductions remain limited rather than producing a capability cliff. The LoCoMo overall score and the LoCoMo subset of NextMem both decrease across all sizes, indicating measurable sensitivity to source composition. However, LongMemEval and the NextMem average do not follow the same pattern, and both of them show improvement for Metis-27B. A related contrast appears between the Metis test set and MemOps. They move in opposite directions, with the direction of the contrast reversing across model sizes.

Table 9 Results of the source-exclusion study. Parentheses give percentage-point differences from the Metis results (bold: positive; gray: negative). 6-Bench Avg is the unweighted mean of LoCoMo, NextMem Avg, Metis Test, MemOps, ATM, and MemDaily.

Model	LoCoMo	NextMem			Metis Test	MemOps	ATM	MemDaily	6-Bench Avg
		LongMemEval	LoCoMo	Avg.					
Metis-4B	13.18 (-3.13)	29.71 (-9.65)	42.46 (-8.02)	37.30 (-4.39)	63.26 (+**6.54**)	16.24 (-1.60)	12.54 (+**2.32**)	42.85 (-9.69)	30.90 (-1.66)
Metis-9B	15.72 (-1.09)	32.00 (-1.36)	43.06 (-8.50)	41.54 (-1.85)	66.93 (+**9.01**)	19.30 (-0.33)	11.65 (-4.84)	41.84 (-5.45)	32.83 (-0.76)
Metis-27B	21.15 (-5.59)	44.21 (+**4.50**)	52.03 (-8.38)	53.31 (+**2.49**)	67.66 (-6.11)	30.89 (+**6.13**)	12.44 (-6.12)	52.10 (-6.94)	39.59 (-2.69)

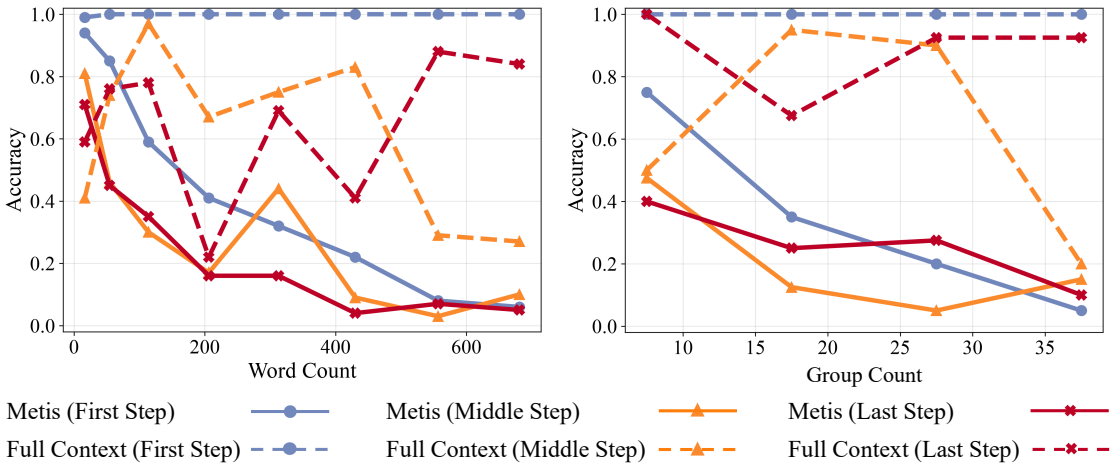

Figure 3 Results of memory capacity at the step-level (left) and trajectory-level (right).

Taken together, these results suggest that excluding these sources preserves most of the overall memory capability while redistributing performance across benchmarks, rather than causing uniform degradation or improvement. In addition, the average across the two OOD benchmarks, ATM and MemDaily, decreases at every model size, although the two benchmarks do not change uniformly.

These changes may be explained from two perspectives. From the perspective of memory as a prediction problem, narrower entity coverage may make the model less sensitive to information involving unseen or low-frequency entities and to how that information may be used in future interactions. From the training-data perspective, a small but concentrated exclusion can produce a disproportionate change in task proportions, shifting the relative supervision across memory behaviors and potentially producing different performance trade-offs across benchmarks.

6.6 Memory Capacity Studies

In this part, we further explore the long-term memory capability of Metis. We evaluate this capability by modeling memory capacity, which comprises step-level and trajectory-level capacity. Specifically, step-level capacity refers to the maximum number of tokens accommodated within a single update. Similarly, trajectory-level capacity denotes the maximum number of update steps within an interaction trajectory.

We construct a testing dataset, which contains 20 fictional users defined over a shared schema of 40 distinct and atomic persona domains. First, the model generates 40 fine-grained domains such as demographics, education and relationships. Then, for each user, the domains are shuffled with a fixed seed and instantiated sequentially. Previously generated attributes are provided as an immutable context to ensure logical consistency within each persona. Each attribute is initially expanded into a biography-style sentence conditioned on the complete persona. After that, one direct question is generated for each domain and reused across all users. Finally, each biography-style message is rewritten as a concise first-person statement that expresses only the corresponding fact. During evaluation, the 40 records of each user form an ordered trajectory. The statements are sequentially stored in memory, and the queries are used to probe the model. In addition, the user attributes serve as the gold answer for LLM-based judging. Based on this dataset, we compare the performance of Qwen3.5-4B with full context and Metis-4B under two evaluation settings.

Step-level Capacity. This setting evaluates how much information the model can encode within a single memory

Table 10 Results of general capability tasks. The gap is calculated as the performance of Metis-4B minus that of Qwen3.5-4B.

Benchmark	Initial Stage			Active Stage		
	Qwen3.5-4B	Metis-4B	Gap	Qwen3.5-4B	Metis-4B	Gap
MMLU-Pro	46.00	45.20	−0.80	46.00	40.90	−5.10
IFEval	79.30	79.85	+0.55	76.71	54.53	−22.18
GSM8K	83.09	82.03	−1.06	84.53	78.92	−5.61
MMMLU	61.30	60.50	−0.80	59.90	56.60	−3.30

update. For each user of step t, the memory state is reset, and the first t statements are concatenated and updated in one operation. The model is then queried about the first, middle, and last facts in the updated content. Since the full history is encoded from scratch at each step, this setting isolates the capacity of a single update as the input length increases. The results are presented in **Figure 3 (left)**.

For the step-level setting, Metis performs well when a single update contains only a small amount of information, but its accuracy decreases rapidly as the input becomes longer. The performance on the first fact shows the clearest downward trend, while the middle and last facts exhibit larger fluctuations. When the input exceeds several hundred words, performance at all three positions becomes low. In contrast, the full-context baseline remains much stronger, especially for the first fact.

Trajectory-level Capacity. This setting evaluates how much information the model can retain over a sequence of memory updates. For each user, the memory state is reset only once and then accumulates throughout the trajectory. The 40 statements are divided into consecutive groups of g statements, with $g = 5$ by default, and each group is concatenated and updated in Metis sequentially. After every group-level update, the model is queried about the first, middle, and current updated facts. This setting measures the capacity of the evolving memory state under repeated updates, with performance reported against the number of updated trajectory steps. The results are presented in **Figure 3 (right)**.

For the trajectory-level setting, performance also declines as the number of updated steps increases. The accuracy of the first fact decreases almost continuously, showing that early information is gradually weakened by later updates. The middle and most recent facts also remain unstable, which suggests that new updates introduce interference throughout the whole memory state rather than only overwriting the oldest information. Although the amount of information in each update is fixed, performance drops clearly as the trajectory becomes longer. This confirms that repeated state transitions and accumulated compression errors form another major limitation of native memory.

6.7 General Capability Studies

Previous experiments have demonstrated the effectiveness of Metis on memory-related tasks. However, integrating native memory into the forward computation potentially influences the backbone's original behavior, which possibly decreases its general capabilities. In this part, we further explore how Metis performs on the general tasks compared with its original backbone. We compare Metis-4B with Qwen3.5-4B and report the performance difference between them. Specifically, we design two settings to evaluate the general capabilities of Metis in different stages. The first is the Initial Stage, which measures performance on general tasks before any information is stored in memory, corresponding to step $t = 1$. In this setting, Metis is reset to an empty memory state before receiving the original prompt in benchmarks, and the backbone receives the same prompt. The second is the Active Stage, which evaluates Metis after it has accumulated irrelevant information over previous interaction steps, where we have the step $t > 1$. For Metis, we reset the memory state and store task-irrelevant messages before providing the benchmark prompt. For the backbone, we prepend the same messages to its prompt. Our experiments are conducted under MMLU-Pro [66], IFEval [67], GSM8K [68], and MMMLU [69]. In IFEval, we adopt the strict evaluation setting, which verifies instruction compliance directly on the original model response without applying the response transformations used by the loose criterion. The detailed prompts of irrelevant messages are provided in **Appendix F.4**. We present the results in **Table 10**.

The results show that Metis largely preserves the general capabilities of its original backbone at the initial stage. It shows only minor decreases on the other tasks. This indicates that the added memory architecture and memory-specific training do not substantially change the model's behavior when the memory state is empty. A different trend appears at the active stage. After irrelevant information is stored, Metis shows consistent performance drops across all benchmarks. The degradation is moderate on MMLU-Pro, GSM8K, and MMMLU, but is much larger on IFEval. This suggests

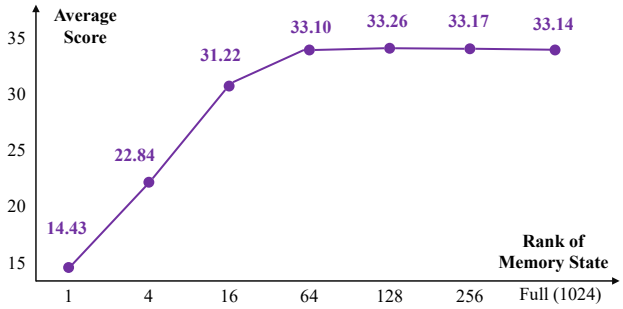

Figure 4 Results of low-rank decomposition under different ranks of memory states.

Table 11 Results of low-rank decomposition across different datasets. Values in parentheses show recovery relative to Full, and the values above 100% are treated as judge variation rather than improvements.

Dataset	$k = 1$	$k = 4$	$k = 16$	$k = 64$	$k = 128$	$k = 256$	Full
LoCoMo (Gold)	11.31 (69.4%)	10.81 (66.3%)	14.21 (87.1%)	16.00 (98.1%)	16.36 (100.3%)	16.14 (99.0%)	16.31 (100.0%)
NextMem	22.80 (54.7%)	27.83 (66.7%)	38.69 (92.8%)	41.43 (99.4%)	41.78 (100.2%)	41.71 (100.0%)	41.69 (100.0%)
Metis Test	17.99 (31.7%)	41.26 (72.7%)	53.64 (94.6%)	56.63 (99.8%)	56.75 (100.1%)	56.04 (98.8%)	56.72 (100.0%)
MemOps (Gold)	5.60 (31.4%)	11.49 (64.4%)	18.36 (102.9%)	18.36 (102.9%)	18.17 (101.8%)	18.79 (105.3%)	17.84 (100.0%)
Overall	14.43 (43.5%)	22.84 (68.9%)	31.22 (94.2%)	33.10 (99.9%)	33.26 (100.4%)	33.17 (100.1%)	33.14 (100.0%)

that irrelevant native memory may introduce noise into the forward computation and interfere with the processing of the current general task, especially on strict instruction following. Overall, Metis retains most of its original general capability before memory is activated, but drops as more information is stored in the memory states.

6.8 Low-rank Decomposition

Storage overhead is a key efficiency metric for memory. In short-term tasks with limited information, parametric memory representations can be further compressed. Therefore, to explore the storage optimization potential of Metis, we apply low-rank decomposition to the memory states for efficient storage and reconstruct them before memory utilization. Specifically, we cast the memory state $\mathbf{M}_t^{(l)}$ to FP32 and compute an SVD along its last two dimensions. Therefore, for the retained rank k, the decomposition process can be represented as

$$\mathbf{M}_t^{(l)} = \mathbf{U}_t^{(l)} \mathbf{\Sigma}_t^{(l)} \mathbf{V}_t^{(l)\top}.$$

We maintain the low-rank approximation $\hat{\mathbf{U}}_t^{(l)} = \mathbf{U}_{t,:,1:k}^{(l)}$, $\hat{\mathbf{\Sigma}}_t^{(l)} = \mathbf{\Sigma}_{t,1:k,1:k}^{(l)}$, and $\hat{\mathbf{V}}_t^{(l)} = \mathbf{V}_{t,:,1:k}^{(l)}$ instead of the original full-rank factors. Here, $\mathbf{U}_{t,:,1:k}^{(l)}$ and $\mathbf{V}_{t,:,1:k}^{(l)}$ denote the first k columns of $\mathbf{U}_t^{(l)}$ and $\mathbf{V}_t^{(l)}$, respectively, while $\mathbf{\Sigma}_{t,1:k,1:k}^{(l)}$ denotes the leading $k \times k$ diagonal submatrix of $\mathbf{\Sigma}_t^{(l)}$. Then, the reconstruction can be expressed as

$$\hat{\mathbf{M}}_t^{(l)} = \hat{\mathbf{U}}_t^{(l)} \hat{\mathbf{\Sigma}}_t^{(l)} \hat{\mathbf{V}}_t^{(l)\top}.$$

We evaluate $k \in \{1, 4, 16, 64, 128, 256\}$ on the four benchmarks with Metis-4B aligned with **Section 6.2**. We compare these models with low-rank memory states to the original Metis models to characterize the trends of their performance degradation, where the full memory-state dimension is 1024. The results are presented in **Figure 4**, which shows a clear trend as the retained rank increases. Extremely low ranks, such as 1 and 4, cause substantial performance degradation. This indicates that a few singular directions are insufficient to preserve the semantic information stored in the memory states. Performance improves rapidly when the rank increases to 16 and becomes close to the full-rank model at rank 64. Further increasing the rank from 64 to 256 brings almost no additional improvement. These results suggest that useful information in the memory states is mainly concentrated in a relatively low-dimensional subspace.

Table 11 further shows the sensitivity to low-rank decomposition in different datasets. The results show that different benchmarks have different levels of sensitivity to aggressive low-rank compression. At very small ranks, performance drops clearly on all datasets. The decrease is particularly large on the Metis test set and MemOps (Gold), suggesting that memory operation tasks require sufficient representational capacity to preserve operation-related information.

Figure 5 Case studies of Metis on different conversational scenarios.

LoCoMo (Gold) is relatively less sensitive at $k = 1$, although its performance fluctuates at very small ranks. This may indicate that some useful conversational information is concentrated in a small number of dominant directions. However, these directions alone are not sufficient to support stable memory utilization. MemOps (Gold) recovers its full-rank performance at $k = 16$ and remains stable at larger ranks. In contrast, the Metis test set continues to improve from $k = 16$ to $k = 64$. This difference suggests that the two memory operation benchmarks require different levels of memory capacity. Across all datasets, performance becomes close to the full model at $k = 64$. The overall recovery reaches 99.9%, while further increasing the rank brings little additional benefit. These results confirm that the memory states still contain substantial redundancy and that most useful information lies in a relatively low-dimensional subspace. They also show that the appropriate compression level depends on the task, since different benchmarks require different amounts and types of memory information.

6.9 Case Studies

We conduct qualitative case studies to show the behavior of Metis-4B under different scenarios. At each turn, the model first generates a response based on the current input and existing memory, and then updates the user input in the memory state. We present several representative cases in **Figure 5** to qualitatively examine the native memory behaviors of Metis-4B. Each case starts from an empty native memory state, and the model must use information stored in previous interaction steps. In the remembering case, Metis correctly stores Alice's food preference and retrieves it in a later query. In the multi-fact case, the model retains several attributes about Alice and correctly selects her age after other attributes are introduced. This result suggests that Metis can bind different values to their corresponding attributes and reduce interference among related facts. The distractor case further shows that an unrelated dialogue turn does not overwrite the stored preference. Metis can therefore distinguish useful memory from ordinary conversational content.

The forgetting case demonstrates that the native memory state is not append-only. After receiving a forgetting instruction, Metis no longer provides the removed preference in the subsequent query. This indicates that the model can modify its latent memory state according to the semantic intent of an instruction. However, the immediate response to the forgetting instruction still repeats the old fact instead of explicitly confirming its removal. The final memory state is correct, but the response at the operation step is not fully aligned with the intended memory operation. This behavior may result from the current step over-emphasizing previous memory states. Overall, these cases show the effectiveness of Metis, while also revealing room for improvement in its consistency.

7 Related Work

7.1 Memory of LLMs and Agents

In recent years, large foundation models and agents have been widely applied to fields such as personal assistants [60, 65], deep research [70–72], and coding agents [3–5]. A critical capability of these systems is memory, which stores past information to support future inference [9]. Based on their representation forms, memory mechanisms of large foundation models and agents are generally categorized into three types, including textual memory, latent memory, and parametric memory [9, 10]. Textual memory typically represents information as text, relying on RAG for storage and retrieval. These methods provide information for backbones to support inference by In-Context Learning (ICL) [73]. For example, MemoryBank [11] proposes a hierarchical storage approach with dual-tower dense retrieval to maintain historical conversations with users. MemTree [74] designs a tree-structured memory mechanism to model the abstraction levels of information, which dynamically updates based on semantic embeddings. In contrast, latent memory captures memory through intermediate activations of models. For example, NextMem [56] compresses factual memory into latent representations through an autoregressive autoencoder, while MemGen [21] generates latent memory tokens that are interwoven into the reasoning process. Additionally, parametric memory injects knowledge into internal model parameters. For example, Locas [75] views the FFN as a soft look-up table. By adding a bypass FFN, it stores test-time information from a key-value perspective. Furthermore, knowledge editing can also be considered a parametric memory method [9]. ROME [50] treats the projection matrix as an associative memory and inserts a new factual association through a rank-one update. Although textual memory remains the most effective approach in industry, latent memory and parametric memory are emerging as promising research directions.

7.2 Fast Weight Programming

Recently, FWP has attracted widespread attention. This paradigm not only uses parameters learned during training (*i.e.,* slow weights), but also maintains dynamic parameters (*i.e.,* fast weights) during inference to capture sequence-dependent information [18]. Existing methods in this line of work generally follow several main directions. Linear attention replaces the softmax kernel with feature maps to achieve linear complexity and a recurrent state [76]. In addition, it has been shown that linear transformers are secretly fast weight programmers [77]. Subsequent works enrich the update rule, such as RetNet [78] and RWKV [79]. Furthermore, state space models compress a sequence into a fixed-size recurrent state with linear-time computation, such as S4 [80] and Mamba [81], while Mamba-2 [82] further reveals a duality between state space models and attention. TTT also treats the recurrent state as fast weights that are optimized by self-supervised gradient descent during inference, such as the TTT layer [83] and Titans [84].

7.3 Memory-Augmented Neural Networks

MANNs introduce explicit memory modules to improve a model's ability to store and retrieve task-specific information during inference. Early work, such as Memory Networks [85] and Neural Turing Machines [19], augments neural controllers with external memory and learns differentiable read and write operations over memory slots. These methods show that neural models can use non-parametric memory to support associative recall, algorithmic reasoning, and few-shot adaptation. However, their memory is usually maintained as a separate storage module, and the memory procedures are often designed independently from the backbone computation. Recent models also maintain dynamic states during inference, such as recurrent memory [86]. Unlike static model parameters learned during training, these dynamic states are updated according to the current input sequence and capture information that changes over time. Our work follows this general direction, but focuses on integrating memory storage and utilization directly into the model computation, so that the model can maintain sequence-dependent information more natively.

8 Conclusion

In this paper, we introduce memory foundation models and provide formal definitions of native memory based on the memory state and memory procedures. Based on this formulation, we propose Metis, the first prototype of memory foundation models. We introduce Metis blocks composed of local memory blocks and hyper memory blocks, enabling the model to maintain compact dense memory states across interaction steps and to update them according to the current input and generated response. We further construct a memory-specific dataset from public benchmarks and design a

mid-training framework with memory reconstruction, memory operation, and regularization objectives. Our experiments verify the effectiveness of Metis and analyze its behavior from multiple perspectives.

Despite these promising results, Metis is still an early step toward memory foundation models. Since the current native memory state compresses information into fixed-size latent parameters, performance may degrade in extremely long-term scenarios, and semantically similar facts may sometimes be confused in the latent space. Therefore, native memory still cannot be viewed as a complete replacement for external memory. Instead, we believe it opens a complementary direction for building future foundation models with more efficient, optimizable, and deeply integrated memory capabilities. Future work may further improve memory capacity, controllability, and interpretability, explore hybrid systems that combine native and external memory, and scale native memory training to broader domains and longer interactions.

References

[1] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 1(2):1–124, 2023.

[2] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. arXiv preprint arXiv:2402.06196, 2024.

[3] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation. ACM Transactions on Software Engineering and Methodology, 35(2):1–72, 2026.

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

[5] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950, 2023.

[6] Fengli Xu, Qianyue Hao, Chenyang Shao, Zefang Zong, Yu Li, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, et al. Toward large reasoning models: A survey of reinforced reasoning with large language models. Patterns, 6(10), 2025.

[7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

[9] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. ACM Transactions on Information Systems, 43(6):1–47, 2025.

[10] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. arXiv preprint arXiv:2512.13564, 2025.

[11] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In Proceedings of the AAAI conference on artificial intelligence, volume 38, pages 19724–19731, 2024.

[12] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. Advances in Neural Information Processing Systems, 38:17577–17604, 2026.

[13] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024.

[14] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, et al. Memos: A memory os for ai system. arXiv preprint arXiv:2507.03724, 2025.

[15] Zeyu Zhang, Quanyu Dai, Rui Li, Xiaohe Bo, Xu Chen, and Zhenhua Dong. Learn to memorize: Optimizing llm-based agents with adaptive memory framework. arXiv preprint arXiv:2508.16629, 2025.

[16] Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. arXiv preprint arXiv:2601.03192, 2026.

[17] Ziliang Guo, Ziheng Li, Bo Tang, Feiyu Xiong, and Zhiyu Li. Memfactory: Unified inference & training framework for agent memory. arXiv preprint arXiv:2603.29493, 2026.

[18] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. Advances in neural information processing systems, 29, 2016.

[19] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014.

[20] Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Di He, Wenhao Huang, and Tianle Cai. In-place test-time training. arXiv preprint arXiv:2604.06169, 2026.

[21] Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. arXiv preprint arXiv:2509.24704, 2025.

[22] Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. δ-mem: Efficient online memory for large language models. arXiv preprint arXiv:2605.12357, 2026.

[23] Ryan Wei Heng Quek, Sanghyuk Lee, Alfred Wei Lun Leong, Arun Verma, Alok Prakash, Nancy F Chen, Bryan Kian Hsiang Low, Daniela Rus, and Armando Solar-Lezama. Memo: Memory as a model. arXiv preprint arXiv:2605.15156, 2026.

[24] Ziwen Xu, Haiwen Hong, Linsong Yu, Benglei Cui, Longtao Huang, Hui Xue, and Ningyu Zhang. How lora remembers? a parametric memory law for llm finetuning. arXiv preprint arXiv:2605.30260, 2026.

[25] Hongkang Yang, Zehao Lin, Wenjin Wang, Hao Wu, Zhiyu Li, Bo Tang, Wenqiang Wei, Jinbo Wang, Zeyun Tang, Shichao Song, Chenyang Xi, Yu Yu, Kai Chen, Feiyu Xiong, Linpeng Tang, and Weinan E. Memory^3: Language modeling with explicit memory. Journal of Machine Learning, 3(3):300–346, 2024.

[26] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In International Conference on Learning Representations, volume 2025, pages 29687–29707, 2025.

[27] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13851–13870, 2024.

[28] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. arXiv preprint arXiv:2410.10813, 2024.

[29] Gregory Kamradt. Needle in a haystack - pressure testing llms. GitHub repository, 2023.

[30] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? arXiv preprint arXiv:2404.06654, 2024.

[31] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers), pages 3119–3137, 2024.

[32] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. ∞bench: Extending long context evaluation beyond 100k tokens. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15262–15277, 2024.

[33] Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 14388–14411, 2024.

[34] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. Advances in Neural Information Processing Systems, 37:106519–106554, 2024.

[35] Zican Dong, Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. Bamboo: A comprehensive benchmark for evaluating long text modeling capacities of large language models. In Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), pages 2086–2099, 2024.

[36] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. Transactions of the Association for Computational Linguistics, 7:453–466, 2019.

[37] Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can context length of open-source llms truly promise? In NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following, 2023.

[38] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), pages 333–342, 2017.

[39] Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. Evaluating the ripple effects of knowledge editing in language models. Transactions of the Association for Computational Linguistics, 12:283–298, 2024.

[40] Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, et al. A comprehensive study of knowledge editing for large language models. arXiv preprint arXiv:2401.01286, 2024.

[41] Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. Temporalwiki: A lifelong benchmark for training and evaluating ever-evolving language models. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 6237–6250, 2022.

[42] Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C Lipton, and J Zico Kolter. Tofu: A task of fictitious unlearning for llms. arXiv preprint arXiv:2401.06121, 2024.

[43] Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, et al. The wmdp benchmark: Measuring and reducing malicious use with unlearning. arXiv preprint arXiv:2403.03218, 2024.

[44] Weijia Shi, Jaechan Lee, Yangsibo Huang, Sadhika Malladi, Jieyu Zhao, Ari Holtzman, Daogao Liu, Luke Zettlemoyer, Noah Smith, and Chiyuan Zhang. Muse: Machine unlearning six-way evaluation for language models. In International Conference on Learning Representations, volume 2025, pages 27797–27818, 2025.

[45] Pengfei Cao, Chenhao Wang, Zhitao He, Hongbang Yuan, Jiachun Li, Yubo Chen, Kang Liu, Jun Zhao, et al. Rwku: Benchmarking real-world knowledge unlearning for large language models. Advances in Neural Information Processing Systems, 37:98213–98263, 2024.

[46] R Eldan and M Russinovich. Who's harry potter? approximate unlearning in llms, arxiv. arXiv preprint arXiv:2310.02238, 2023.

[47] Shengyuan Hu, Neil Kale, Pratiksha Thaker, Yiwei Fu, Steven Wu, and Virginia Smith. BLUR: A benchmark for LLM unlearning robust to forget-retain overlap, 2026.

[48] Naman Deep Singh, Maximilian Müller, Francesco Croce, and Matthias Hein. Unlearning that lasts: Utility-preserving, robust, and almost irreversible forgetting in llms. arXiv preprint arXiv:2509.02820, 2025.

[49] Alexey Dontsov, Dmitrii Korzh, Alexey Zhavoronkin, Boris Mikheev, Denis Bobkov, Aibek Alanov, Oleg Rogov, Ivan Oseledets, and Elena Tutubalina. Clear: Character unlearning in textual and visual modalities. In Findings of the Association for Computational Linguistics: ACL 2025, pages 20582–20603, 2025.

[50] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. Advances in neural information processing systems, 35:17359–17372, 2022.

[51] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. Transactions of the Association for Computational Linguistics, 10:539–554, 2022.

[52] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. Transactions of the Association for Computational Linguistics, 9:346–361, 2021.

[53] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Findings of the Association for Computational Linguistics: EMNLP 2023, pages 5687–5711, 2023.

[54] Zehao Lin, Xixuan Hao, Renyu Fu, Shaobo Cui, Kai Chen, Chunyu Li, Zhiyu Li, and Feiyu Xiong. A survey on long-term memory security in llm agents: Attacks, defenses, and governance across the memory lifecycle, 2026.

[55] Xixuan Hao, Zeyu Zhang, Zehao Lin, Yihang Sun, Ziliang Guo, Xichong Zhang, Yuxuan Liang, Feiyu Xiong, and Zhiyu Li. Memops: Benchmarking lifecycle memory operations in long-horizon conversations, 2026.

[56] Zeyu Zhang, Rui Li, Xiaoyan Zhao, Yang Zhang, Wenjie Wang, Xu Chen, and Tat-Seng Chua. Nextmem: Towards latent factual memory for llm-based agents. arXiv preprint arXiv:2603.15634, 2026.

[57] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In Proceedings of the 2016 conference on empirical methods in natural language processing, pages 2383–2392, 2016.

[58] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Proceedings of the 2018 conference on empirical methods in natural language processing, pages 2369–2380, 2018.

[59] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026.

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems, 33:9459–9474, 2020.

[61] Yan Wang, Dongyang Ma, and Deng Cai. With greater text comes greater necessity: Inference-time training helps long text generation. arXiv preprint arXiv:2401.11504, 2024.

[62] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[63] Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. arXiv preprint arXiv:2607.02770, 2026.

[64] Jingbiao Mei, Jinghong Chen, Guangyu Yang, Xinyu Hou, Margaret Li, and Bill Byrne. According to me: Long-term personalized referential memory qa. arXiv preprint arXiv:2603.01990, 2026.

[65] Zeyu Zhang, Quanyu Dai, Luyu Chen, Zeren Jiang, Rui Li, Jieming Zhu, Xu Chen, Yi Xie, Zhenhua Dong, and Ji-Rong Wen. Memsim: A bayesian simulator for evaluating memory of llm-based personal assistants. Advances in Neural Information Processing Systems, 38:90475–90511, 2026.

[66] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024.

[67] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. arXiv preprint arXiv:2311.07911, 2023.

[68] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

[69] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

[70] Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. Deep research agents: A systematic examination and roadmap. arXiv preprint arXiv:2506.18096, 2025.

[71] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 414–431, 2025.

[72] Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. arXiv preprint arXiv:2506.11763, 2025.

[73] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In Proceedings of the 2024 conference on empirical methods in natural language processing, pages 1107–1128, 2024.

[74] Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In International Conference on Learning Representations, volume 2025, pages 990–1023, 2025.

[75] Sidi Lu, Zhenwen Liang, Dongyang Ma, Yan Wang, Haitao Mi, and Dong Yu. Locas: Your models are principled initializers of locally-supported parametric memories. arXiv preprint arXiv:2602.05085, 2026.

[76] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In International conference on machine learning, pages 5156–5165. PMLR, 2020.

[77] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In International conference on machine learning, pages 9355–9366. PMLR, 2021.

[78] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. arXiv preprint arXiv:2307.08621, 2023.

[79] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, et al. Rwkv: Reinventing rnns for the transformer era. In Findings of the association for computational linguistics: EMNLP 2023, pages 14048–14077, 2023.

[80] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. arXiv preprint arXiv:2111.00396, 2021.

[81] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.

[82] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. arXiv preprint arXiv:2405.21060, 2024.

[83] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. arXiv preprint arXiv:2407.04620, 2024.

[84] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. Advances in Neural Information Processing Systems, 38:113506–113543, 2026.

[85] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. arXiv preprint arXiv:1410.3916, 2014.

[86] Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. Advances in Neural Information Processing Systems, 35:11079–11091, 2022.

[87] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. Proceedings of Machine Learning and Systems, 6:1–13, 2024.

[88] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. S-lora: Serving thousands of concurrent lora adapters. arXiv preprint arXiv:2311.03285, 2023.

A Roadmap for Memory Foundation Models

Metis represents an initial exploration of memory foundation models. It transforms memory from an external information-management module into a persistent internal state, integrating memory storage and utilization directly into the forward computation of the model. The significance of native memory, however, extends beyond improving information retention capability. In the longer term, it may reshape the computational paradigm, learning process, cognitive structure, and capability development of foundation models. As illustrated in **Figure 6**, we envision five progressive levels of capabilities in the development of memory foundation models: **stateful capability**, **self-managing capability**, **experience-learning capability**, **persistent cognitive capability**, and **self-evolving capability**. These capabilities characterize how memory may become progressively integrated into the foundation model itself, progressing from persistent state to autonomous memory organization, experience-driven learning, persistent cognition, and continual capability evolution.

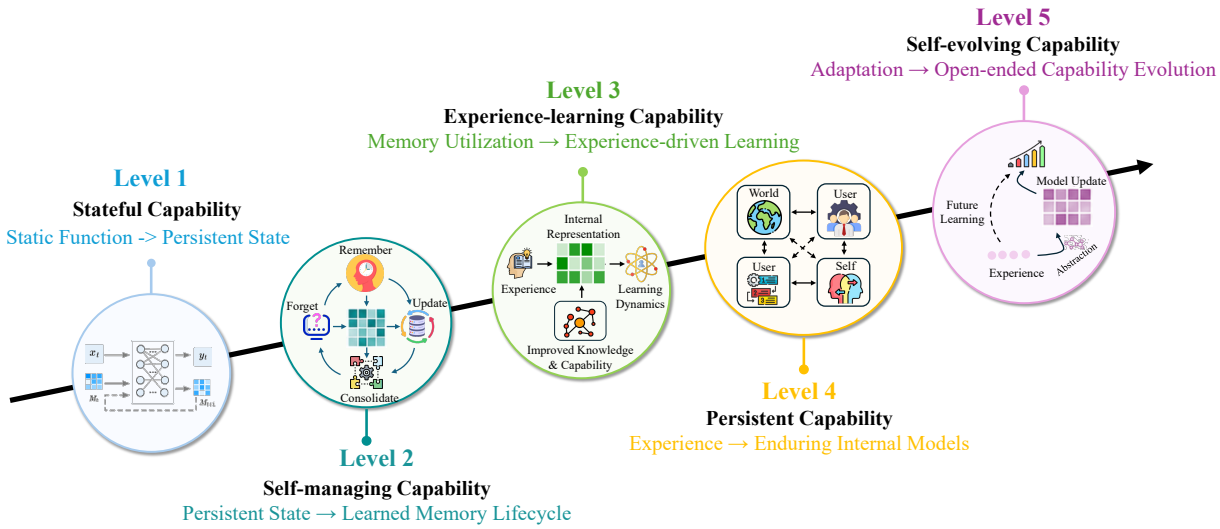

Figure 6 Roadmap for memory foundation models. Native memory aims to progressively transform foundation models from stateless predictors into stateful learners and, ultimately, into models capable of continual self-evolution. The five capabilities represent increasingly deep changes to the model's computation, memory organization, learning process, cognitive representations, and capability formation. Their development is supported by advances in memory architecture, learning objectives, scalable training, interpretability and control, and long-horizon evaluation.

Level I: Stateful Capability. Most existing foundation models remain fundamentally stateless conditional predictors. At each inference step, their outputs are determined by fixed model parameters and the current context, while continuity across interactions is primarily maintained by repeatedly supplying historical information as external context. Therefore, the first level is a transition from a **static function to a persistent state**. A memory foundation model maintains a dynamic internal state across inference steps. Its output is jointly determined by the current input and its previous state, while each interaction updates the state used in subsequent computation:

$$(Y_t, \mathbf{M}_{t+1}) = f_\Phi(X_t, \mathbf{M}_t),$$

where Φ denotes the static model parameters and $\mathbf{M}_t$ denotes the native memory state at step t. Unlike textual context supplied from outside the model, $\mathbf{M}_t$ is directly coupled with the model's internal computation and evolves continuously during interaction. This transition changes the basic computational unit of a foundation model. The model is no longer merely a static mapping from inputs to outputs, but rather a stateful computational system that evolves over time.

Level II: Self-Managing Capability. Possessing a persistent state does not by itself constitute a complete memory capability. Information observed in real environments differs in importance, validity, abstraction level, time scale, and security requirements. A model must therefore learn how to organize and maintain its internal state autonomously. The second level moves from **persistent state to a learned memory lifecycle**. The model should determine what to remember, update, consolidate, and forget according to the semantics of incoming information, its expected future utility,

and applicable privacy and safety constraints. At this level, memory operations are no longer implemented primarily through external rules or discrete workflows. Instead, both the memory state and the procedures that transform it become native, trainable components of the foundation model. This enables memory to be selected, organized, and evolved within the model's continuous computational space.

Level III: Experience-Learning Capability. Once a model can autonomously maintain memory, the role of memory can expand from information support to capability adaptation. The model should not only remember what happened, but also change as a consequence of what it has experienced. The third level marks a transition from **memory utilization to experience-driven learning**. Interaction histories become reward signals that the model can use to refine representations, knowledge, and behavioral regularities. Experiences involving success, failure, feedback, or environmental change can be transformed into reusable internal capabilities rather than remaining isolated records. This direction may gradually connect pre-training, in-context learning, test-time adaptation, and continual learning within a unified framework. Instead of remaining completely fixed after training, a foundation model could continuously adapt to new users, tasks, and environments while preserving previously acquired capabilities.

Level IV: Persistent Cognitive Capability. Experience learning explains how a model may adapt through interaction, but more advanced intelligence requires the formation of structured, persistent, and continually updated internal cognition. The next level moves from **accumulated experience to enduring internal models**. A foundation model should maintain evolving representations of the world, users, tasks, time, and itself. These representations should capture temporal changes, causal dependencies, uncertainty, and conflicts between new evidence and existing beliefs. When the environment changes, the model should be able to revise its internal representations while preserving global consistency. At this level, memory no longer consists of disconnected pieces of historical information. It becomes the substrate through which the model maintains cognitive continuity over extended periods. Planning, personalization, and complex decision-making may emerge as downstream expressions of this capability, but the defining transformation occurs within the foundation model's internal cognitive representations rather than in an external agent workflow.

Level V: Self-Evolving Capability. The long-term objective of memory foundation models is to convert accumulated experience into the continual development of the model's own capabilities. A model should not only adapt its current state, but also reflect on, abstract, and reorganize past experience to discover new knowledge structures and learning strategies. The final level represents a transition from **local adaptation to open-ended capability evolution**. The model identifies experiences with long-term value, abstracts transferable knowledge from specific interactions, and incorporates the resulting insights into future learning. Then, the exploration and exploitation will constitute a continual feedback loop between memory foundation models and the environment.

Outlook. Memory foundation models should not be viewed as conventional foundation models equipped with a stronger storage module. They point toward a more fundamental change in the paradigm of the foundation model. Under this view, memory may become a foundational mechanism connecting computation, learning, cognition, and continual self-evolution.

> **A POSSIBLE PARADIGM SHIFT**
>
> Foundation models may evolve from **stateless predictors** into **stateful learners**, and ultimately into learning systems that autonomously organize memory, maintain persistent cognition, and develop new capabilities through accumulated experience.

B Extensive Experiment Results

The results of memory operation tasks on MemOps (Full) are presented in **Table 12**. According to the results, access to complete textual evidence remains the strongest setting. Full-context models achieve consistently high performance across all four operations, with moderate overall gains from increasing the backbone size. In contrast, performance drops sharply when only partial context is available. Standard Qwen models obtain near-zero scores without context, confirming that these operations cannot be performed reliably using backbone knowledge alone. The partial-context results also show that increasing model size cannot compensate for missing historical evidence.

Among no-context methods, Metis-27B achieves the best overall performance. It obtains the highest scores on updating

Table 12 The performance on memory operation tasks under MemOps (Full). Full-context and partial-context results are shown in gray to visually distinguish context-access settings from the no-context comparison. Within the No Context setting, the best and second-best scores are **bolded** and underlined, respectively. Average represents the micro-average performance.

Type	Method	Remember	Update	Forget	Reflect	Average
Full Context	Qwen3.5-4B	87.05	85.42	81.36	79.66	83.52
	Qwen3.5-9B	89.58	89.58	77.95	84.14	85.69
	Qwen3.5-27B	91.52	92.13	81.82	82.59	87.19
Partial Context	Qwen3.5-4B	31.55	23.15	20.23	14.48	22.83
	Qwen3.5-9B	25.30	12.96	15.23	7.24	15.77
	Qwen3.5-27B	28.42	15.28	17.50	10.17	18.50
No Context	Qwen3.5-4B	3.57	0.00	1.82	0.00	1.51
	Qwen3.5-9B	3.57	1.85	0.68	0.00	1.65
	Qwen3.5-27B	4.17	0.93	0.68	0.69	1.84
	Temp-LoRA-4B	14.88	14.35	2.95	6.38	9.98
	Temp-LoRA-9B	<u>19.79</u>	14.58	4.09	7.76	<u>12.19</u>
	Temp-LoRA-27B	**20.68**	10.88	2.50	5.69	10.83
	δ-Mem	4.46	5.09	1.36	1.21	3.06
	Metis-4B	10.42	13.66	<u>4.77</u>	<u>9.66</u>	9.70
	Metis-9B	13.84	<u>14.81</u>	**7.27**	<u>9.66</u>	11.53
	Metis-27B	17.26	**17.13**	3.86	**20.52**	**15.35**

and reflection, while Metis-9B performs best on forgetting. In particular, the substantial improvement on reflection suggests that native memory can support higher-level reasoning over stored information rather than only preserving individual facts. Temp-LoRA remains strongest on remembering, indicating that temporary parameter adaptation is effective for direct information retention. However, Metis provides a more balanced advantage across memory operations and outperforms Temp-LoRA in the overall average.

Scaling Metis from 4B to 9B produces only moderate improvements, whereas Metis-27B substantially increases the average score. This demonstrates that a larger backbone can strengthen native memory operations when sufficient model capacity is available. However, the gains are not uniform across operations. For example, Metis-27B improves updating and reflection but performs worse than Metis-9B on forgetting. This suggests that different memory operations may require different mechanisms and may not benefit equally from backbone scaling. Despite these improvements, a large gap from the full-context setting remains, showing that accurately storing, modifying, and reasoning over complex histories is still challenging.

C Further Analysis of Update Designs

C.1 LU and GDU across Model Scales

As discussed in **Section 6.3**, using a linear update (LU) to replace the GDN-based update (GDU) yields competitive performance at the 4B scale. In this part, we further explore their performance at the 9B and 27B scales. During the training phase, the Metis-9B LU did not show sharp fluctuations on the validation curve, so it was trained to 14k steps without early stopping. The reported Metis-9B GDU result is based on the 8k checkpoint.

Across all three scales, LU scores higher on the Metis test set but lower on LoCoMo (Gold). This recurring split points to a task-dependent trade-off between direct memory operations and long conversational memory. The aggregate comparison is driven by the Metis test set at 9B and LoCoMo (Gold) at 27B, while the two updates remain nearly tied at 4B. We also find that LU has a sharp drop in LoCoMo (Gold), which may indicate that LU is more vulnerable over long conversational trajectories. In addition, LU consistently performs better on the Metis test set, possibly because its simpler update rule is easier to fit to the memory operations emphasized during training.

Table 13 Performance comparison of Metis with LU and GDU across different model scales. Overall is the equal-weight macro-average score of the four benchmarks.

Scale	Update	LoCoMo (Gold)	NextMem	Metis Test Set	MemOps (Gold)	Overall
4B	GDU	16.31	41.69	56.72	17.84	33.14
	LU	11.97	42.78	58.54	18.50	32.95
9B	GDU	16.81	43.39	57.92	19.63	34.44
	LU	15.37	40.16	68.66	18.17	35.59
27B	GDU	26.74	50.82	73.77	24.76	44.02
	LU	14.16	52.09	75.32	24.44	41.50

Table 14 Data and structure ablations from the LU baseline on Metis-4B. All scores are percentages. Overall is the equal-weight macro-average of the four benchmarks, and ΔAvg. is the relative performance gap from the reported LU reference.

Type	Model	LoCoMo (Gold)	NextMem	Metis Test Set	MemOps (Gold)	Overall	ΔAvg.
Reference	LU full	11.97	42.78	58.54	18.50	32.95	–
Data Ablation	LU w/o MS	16.98	40.37	62.56	14.78	33.67	+2.20%
	LU w/o MS+MP	15.44	37.70	44.84	17.42	28.85	-12.44%
Structure Ablation	LU w/o SA	12.25	19.61	26.85	6.12	16.21	-50.81%
	LU w/o OQ	10.56	32.50	58.25	14.03	28.83	-12.49%
	LU w/o QKN	12.69	36.89	50.35	12.15	28.02	-14.96%

C.2 Ablations from the LU Baseline

The major ablation study in **Section 7** takes the GDU-based Metis-4B as its reference. In this section, we replace the GDU with LU and repeat the same data and structure ablations, testing whether their effects depend on the update rule. All variants are trained for 14,000 steps. We take the LU result reported as w/o GDU in **Table 7** as the reference, and follow the main table in reporting relative changes in Overall.

Table 14 shows that, except for removing MS alone, the LU ablations produce the expected declines. Removing SA causes by far the largest degradation, identifying adaptive aggregation as the most consequential structural component under LU as well. Removing OQ or QKN also hurts Overall, matching the direction of the GDU-based study and showing that their contributions are not specific to GDU. Removing MS alone slightly improves Overall, in contrast to its decrease in the main GDU-based ablation. This sign reversal suggests that the effect of MS depends on the update rule. MS trains the model to maintain and revise multiple entities within a shared memory state. One possible explanation is that GDU's delta-rule update performs a key-conditioned correction that approximately overwrites the target entity's existing value association. This mechanism may make multiple entities easier to manage, whereas LU globally decays the memory state and adds new key-value content, potentially increasing cross-entity interference. Finally, removing both MS and MP produces a clear overall degradation, showing that the auxiliary data remains important as a whole.

D Transfer Across Backbone Families and Scales

Metis can be applied to various compatible causal decoder-only Transformer backbones by integrating its native-memory components into their Transformer layers. Since the main experiments use Qwen3.5 backbones, we test this architectural flexibility by applying Metis to Llama3.1-8B [62], Gemma4-12B [63], Gemma4-31B [63], and Llama3.1-70B [62].

Setup. We replace the Qwen3.5 backbone with each transferred backbone and otherwise follow the same training and evaluation setup as the main experiments, reporting the checkpoint at step 14,000. For each backbone, we compare three conditions. No Context provides no interaction history, Metis uses only its internal memory state without replaying the original context, and Full Context directly provides the complete interaction history. We also retain the Qwen3.5-based Metis-4B and Metis-9B results from the main evaluation as references.

Results. As shown in **Table 15**, all four transferred Metis variants exhibit the same broad pattern. On each benchmark, memory-only performance is higher than the corresponding no-context control but lower than the full-context control. Relative to the Qwen-based Metis-4B and Metis-9B references, some transferred variants improve the overall performance,

Table 15 Backbone-transfer evaluation. Overall is the equal-weight macro-average across the four benchmarks.

Method	LoCoMo (Gold)	NextMem	Metis Test Set	MemOps (Gold)	Overall
Llama3.1-8B (No Context)	0.25	19.27	19.98	3.25	10.69
Llama3.1-8B (Full Context)	62.80	75.75	73.42	70.48	70.61
Gemma4-12B (No Context)	0.07	15.69	16.40	1.27	8.36
Gemma4-12B (Full Context)	66.39	81.05	78.05	83.71	77.30
Gemma4-31B (No Context)	0.00	10.55	10.39	0.66	5.40
Gemma4-31B (Full Context)	70.38	80.79	77.96	83.57	78.18
Llama3.1-70B (No Context)	0.13	26.44	21.24	3.01	12.71
Llama3.1-70B (Full Context)	64.93	77.22	74.56	83.52	75.06
Metis (Llama3.1-8B)	20.56	39.98	72.33	11.25	36.03
Metis (Gemma4-12B)	21.68	49.00	58.77	21.28	37.68
Metis (Gemma4-31B)	17.80	44.98	53.37	17.61	33.44
Metis (Llama3.1-70B)	22.17	52.36	70.95	11.68	39.29
Metis-4B (Qwen3.5-4B)	16.31	41.69	56.72	17.84	33.14
Metis-9B (Qwen3.5-9B)	16.81	43.39	57.92	19.63	34.44

but these gains do not hold across every task.

These results show that Metis's memory-specific mid-training transfers to the tested compatible Gemma and Llama backbones and model scales. However, we find that the performance varies across model sizes and benchmarks without a monotonic scaling trend. One possible reason is that the shared training recipe may not be equally well matched to every backbone and scale. Thus, the observed transferability does not imply backbone-independent behavior, universal superiority, or complete preservation of full-context capability.

E Evaluation Implementation Details

DenseRAG. We construct the retrieval corpus from only the context visible to the current test instance. The context is divided at sentence boundaries, and sentences longer than 256 embedding-model tokens are further split into contiguous chunks. Retrieval is performed by cosine similarity, and the top-5 chunks are provided to the corresponding Qwen3.5-4B, 9B, or 27B generator. Gold answers, evidence identifiers, and future turns are never included in the retrieval corpus.

Temp-LoRA. Following the official repository's raw-text adaptation design, we implement an official-like memory-task adaptation of Temp-LoRA for Qwen3.5-4B, 9B, and 27B. For each test instance, the LoRA and optimizer are reset, and each memory step is capped at 4,096 tokens (enough for reported benchmarks) and split into 1,024-token chunks. We perform two updates per chunk in BF16 with batch size 1 and AdamW (learning rate 5×10^{-5}, zero weight decay, and no scheduler). The LoRA uses rank 64, scaling factor 64, and dropout 0.05, and is applied to the attention and feed-forward projections. Its parameters and optimizer state persist within an instance and are discarded before the next instance.

δ-Mem. We use the officially released `delta-mem_qwen3_4b-instruct` adapter with the Qwen3-4B-Instruct-2507 backbone and the official δ-Mem runtime. Each raw memory step is ingested as a separate user message. Before issuing the question-only query, we retain only the online δ-Mem state and clear the chat history, processed input IDs, and KV cache. We reset δ-Mem before each instance.

Metis. We reset Metis to an empty `LocalMemory` state before each instance. During the query phase, Metis receives only the question prompt.

F Prompts

F.1 Prompt Notation and Coverage

Double braces, such as `{{question}}`, denote values inserted at runtime. Only prompt text is shown verbatim. Per-instance questions, contexts, retrieved chunks, and answers are omitted. Automatic wrapping inside the prompt boxes is typographical only. Ablation and LowRank runs add no natural-language prompt.

F.2 Information and Query Prompts of Baselines

F.2.1 Qwen3.5 Backbone

Query of Memory-based QA Tasks (No-context).

```
Question: {{question}}
Answer with a short phrase. If the answer is not known from the given information, say "No
information available".
```

Query of Memory Operation Tasks (No-context).

```
Question: {{question}}
Answer the question using the memory context. Be concise, but include all necessary details. If
the answer is not known from the given information, say "No information available".
```

Query of Memory-based QA Tasks (Partial-context).

Optional metadata fields are emitted only when available.

```
Retrieved context:
[chunk 1 | date={{optional_date_time}} | session={{optional_session_id}} |
speaker={{optional_speaker}}]
{{retrieved_chunk_text}}

Question: {{question}}
Answer with a short phrase using only the retrieved context. If the answer is not known from the
retrieved context, say "No information available".
Short answer:
```

Query of Memory Operation Tasks (Partial-context).

```
Retrieved context:
[chunk 1]
{{retrieved_chunk_text}}

Question: {{question}}
Answer with a short phrase using only the retrieved context. If the answer is not known from the
retrieved context, say "No information available".
Short answer:
```

Query of LoCoMo (Gold) in Memory-based QA Tasks (Full-context).

```
Evidence-session context:
SESSION: {{session_id}}
DATE: {{date_time}}
{{dialogue_turn_id}} {{speaker}} said: "{{dialogue_text}}" Shared image caption:
{{optional_image_caption}}

Question: {{question}}
Answer with a short phrase. If the answer is not known from the given information, say "No
information available".
```

Query of NextMem in Memory-based QA Tasks (Full-context).

```
Reference context:
{{reference_context}}

Question: {{question}}
Answer with a short phrase. If the answer is not known from the given information, say "No
information available".
```

Query of Memory Operation Tasks (Full-context).

The context block repeats for each normalized context item.

```
Memory context:

{{context_id}}:
{{memory_context}}

Question: {{question}}
Answer the question using the memory context. Be concise, but include all necessary details. If
the answer is not known from the given information, say "No information available".
```

F.2.2 Metis, Temp-LoRA, and δ-Mem

During the information stage, we retain each method's method-specific write interface while keeping the ordered memory-step contents fixed. Metis prepends a fixed, answer-independent commit instruction to each memory step before its memory-commit operation. Temp-LoRA updates its temporary LoRA directly on the memory context. δ-Mem passes each raw memory step as a user message through the officially released runtime's native chat-message ingestion path to update its online state. At query time, all three methods use the same prompt.

Share Query of Metis, Temp-LoRA, and δ-Mem.

```
Answer from the learned memory state produced during the information phase.
Give the shortest factual answer you can. Do not explain.
Question: {{question}}
Short answer:
```

Information-stage Prompt of Metis in Memory-based QA Tasks.

```
Conversation memory segment.
Commit the following dated dialogue segment to memory for later question answering.

{{memory_step.content}}
```

Information-stage Prompt of Metis in Memory Operation Tasks.

```
Conversation memory segment.
Commit the following dialogue segment to memory for later question answering.

{{memory_step.content}}
```

Information-stage Payload of Temp-LoRA and δ-Mem.

These methods pass `memory_steps[*].content` directly during the information step.

```
{{memory_step.content}}
```

F.3 Prompts for LLM-as-a-Judge

Formal scoring uses `gpt-4.1-mini` with temperature 0, three repeats. For each example, the score is the median of the three LLM-as-a-judge scores.

System Message.

```
You are a conservative but fair evaluator for a memory question-answering benchmark.  Your job is
to avoid overly generous partial credit while still accepting truly equivalent answers, aliases,
abbreviations, and harmless formatting differences.  Return JSON only.
```

User Instruction.

```
Grade model_output against gold_answer for the question.  Return JSON with keys:  score (0 to 1),
pass (boolean), matched_points (array of strings), missed_points (array of strings), and
rationale (short string).  Use this strict rubric:  give 1.0 only when the answer contains the
correct core entity/value/date/relationship asked for, allowing aliases and semantically
equivalent wording.  Give 0.5 to 0.75 only when the output includes the correct core answer but
has minor extra wording, minor imprecision, or one secondary omission.  Give 0 for a different
person, organization, place, number, date, title, relation, or answer choice; for a broad
category when the gold answer is a specific entity; for answers that merely share common words
with gold; for plausible guesses unsupported by the exact answer; or when the model says the
answer is unknown/unavailable while gold is answerable.  Do not reward explanation quality if the
final answer is wrong.  If the question asks for a country/state/type and the model gives exactly
that correct country/state/type, it is correct even if it could be guessed from world knowledge.
```

User Payload (JSON Schema).

```
{
  "instruction":  "{{judge_instruction_above}}",
  "question":  "{{question}}",
  "gold_answer":  "{{gold_answer}}",
  "model_output":  "{{model_output}}",
  "raw_category":  "{{raw_category}}",
  "is_adversarial":  "{{is_adversarial}}",
  "baseline":  "{{baseline}}"
}
```

F.4 Prompts in General Capability Study

The active stage setting uses the same message for both models. For Qwen3.5-4B, it is prepended to the complete benchmark prompt. For Metis-4B, the model is reset for each example, the message is stored in the memory state, and the unchanged benchmark prompt is then provided.

```
This session is for general-purpose evaluation.  For any upcoming question, follow the
instruction closely, reason carefully when needed, and answer based on the information available.
```

MMLU-Pro, IFEval, GSM8K, and MMMLU retain their native per-instance benchmark prompts.

G Efficiency

This appendix evaluates the inference efficiency of Metis at the 4B scale, characterizing both the benefits of native memory and the costs it introduces. We first measure application-level end-to-end and query latency on LoCoMo (Gold). Furthermore, we isolate the effect of history through a controlled sweep over 512 to 128K context tokens, with latency decomposed into fine-grained stage-level components. Beyond latency, we further examine the storage efficiency of

Table 16 End-to-end latency, query latency, input length, generation length, and effective throughput on the LoCoMo (Gold) evidence-session setting. The smallest and second-smallest latency values are **bolded** and underlined, respectively.

Method	E2E Latency (s)		Query Latency (s)		Prompt Tokens (Avg./P95)	Committed Tokens (Avg./P95)	Generated Tokens		Effective Throughput (tokens/s)
	Avg.	P95	Avg.	P95			Avg.	P95	
No Context	**0.149**	**0.145**	**0.149**	**0.145**	50.2 / 58.0	–	3.0	3.0	<u>20.16</u>
Full Context	0.607	3.012	0.607	3.012	1410.6 / 3345.8	–	5.7	16.0	9.37
Partial Context	<u>0.268</u>	<u>0.456</u>	<u>0.223</u>	<u>0.412</u>	290.5 / 322.0	–	4.0	11.0	17.73
Temp-LoRA	1.567	3.292	0.305	0.746	56.2 / 64.0	1386.0 / 3365.2	5.2	15.0	17.11
δ-Mem	0.884	1.600	0.656	1.365	49.2 / 57.0	1377.2 / 3353.1	8.8	21.0	13.44
Metis	0.562	0.926	0.360	0.609	56.2 / 64.0	1449.0 / 3515.4	8.3	15.0	**23.15**

maintaining a fixed-size memory state, measuring the storage that each method must persist per session as history grows. Finally, we prototype multi-user serving with a LoRA-based serving architecture, measuring its isolation and efficiency at serving scale.

G.1 End-to-End Latency on LoCoMo (Gold)

Evaluation Protocol. We evaluate all methods on the 1,527 examples in the LoCoMo (Gold) evidence-session setting, following the baseline configurations described in **Section 6**. Each example is evaluated independently under a cold-start protocol: the method-specific state is reset, the evidence session is processed, and the question is then answered without reusing state across examples. All methods use greedy decoding with at most 96 generated tokens. Each run includes one unmeasured warm-up example, and CUDA synchronization is applied at every timing boundary. End-to-end latency includes retrieval, test-time adaptation, and memory writing when required, followed by query generation. Query latency starts after these preparation operations have completed. Model loading and answer evaluation are excluded. Because generation uses natural stopping behavior, the results measure application-level latency rather than controlled decode-only latency.

Results. Partial Context has the lowest end-to-end latency among methods that use historical information, with an average of 0.268 seconds and a P95 of 0.456 seconds. Metis achieves an average latency of 0.562 seconds, slightly below the 0.607 seconds of Full Context, while reducing P95 latency from 3.012 to 0.926 seconds, a reduction of 69.3%. Compared with δ-Mem, Metis reduces average and P95 end-to-end latency by 36.4% and 42.2%, respectively; compared with Temp-LoRA, the corresponding reductions are 64.1% and 71.9%. Metis also reduces the average query latency from 0.656 to 0.360 seconds relative to δ-Mem, with P95 decreasing from 1.365 to 0.609 seconds.

Analysis. The slightly lower average latency and substantially lower P95 latency relative to Full Context indicate that Metis provides a more stable query path across LoCoMo (Gold) instances with varying history lengths. Full Context processes 1,410.6 prompt tokens on average and 3,345.8 tokens at P95 for every query. Metis commits a similar amount of historical information, at 1,449.0 tokens on average and 3,515.4 tokens at P95, but its subsequent query contains only 56.2 prompt tokens on average and 64 tokens at P95. Thus, Metis shifts historical-information processing into a separate memory-write stage and avoids replaying the original evidence during querying, contributing to its lower query and tail latency. Its advantage over δ-Mem comes mainly from memory utilization rather than memory construction: its average write latency is moderately lower at 0.199 versus 0.227 seconds, whereas it reduces average and P95 query latency by 45.2% and 55.4%, respectively. Temp-LoRA, in contrast, is dominated by its 1.186-second test-time adaptation. These comparisons show that Metis has lower online overhead than the two state-based parametric memory baselines. Partial Context remains the fastest cold-start method because its retrieval stage costs only 0.044 seconds on average and produces a compact 290.5-token query prompt. The token statistics of δ-Mem follow its Qwen3 tokenizer, while the other methods use the Qwen3.5 tokenizer.

G.2 Latency Scaling with Context Length

Evaluation Protocol. We evaluate all six methods at the 4B scale on a single NVIDIA A800 GPU using controlled histories from 512 to 128K tokens. Each final query contains 512 content tokens, excluding method-specific templates,

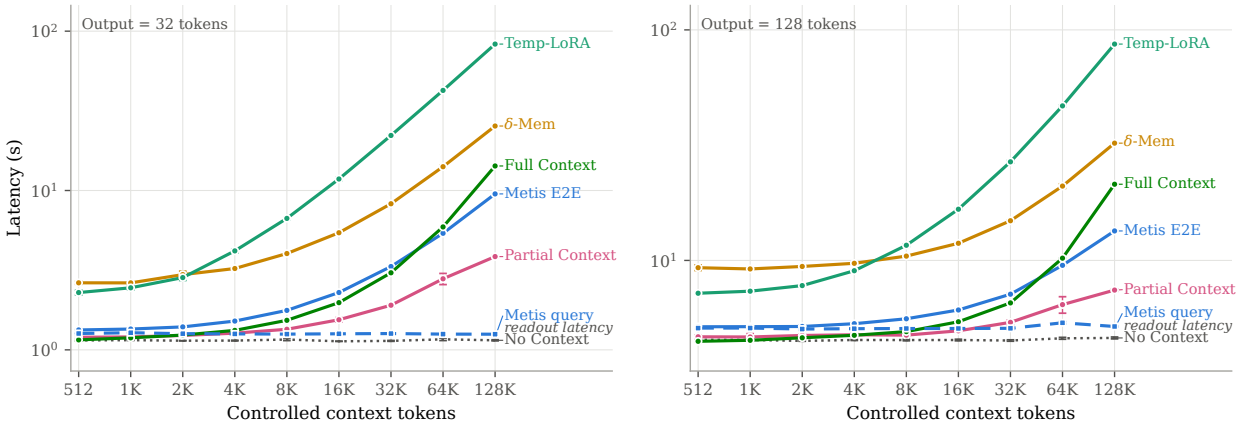

Figure 7 End-to-end latency as controlled context length increases, with 32 generated tokens (left) and 128 generated tokens (right). Metis E2E includes streaming history ingestion and query generation; Metis query excludes ingestion. Readout latency is computed as Metis query latency minus No Context latency. The first measured crossover between Metis E2E and Full Context occurs at 64K for both output lengths.

and generates either 32 or 128 tokens with batch size 1. For every method, context length, and output length, we report the mean and standard deviation over five measured runs after one warm-up run to remove kernel initialization. Metis processes the history in chunks of at most 2K tokens, so its number of commits grows naturally with context length. Partial Context rebuilds its index over the complete history for every run and retrieves the top five sentence chunks. This is a generation-only experiment without answer scoring. End-to-end latency includes all method-specific history preparation or writing and the subsequent query, whereas query latency is measured after history preparation has completed.

Results. At context lengths up to 32K, Metis has slightly higher end-to-end latency than Full Context. The ordering reverses at the first measured 64K point for both output lengths: for 32 generated tokens, Full Context and Metis take 5.909 and 5.390 seconds, respectively, giving a 1.096× speedup; for 128 generated tokens, they take 10.228 and 9.522 seconds, giving a 1.074× speedup. At 128K, these E2E speedups increase to 1.497× (14.254 versus 9.521 seconds) and 1.595× (21.404 versus 13.423 seconds), respectively, in **Figure 7**.

Figure 8 further shows that pure memory commit is a small fraction of Metis latency. At 64K and 128K, commit takes approximately 0.203 and 0.406 seconds, respectively, and accounts for only 2.1–4.3% of Metis E2E latency across the two output lengths; most ingestion time instead comes from the history-encoding backbone forwards, which grow from approximately 3.86 to 7.65 seconds. Metis also has lower E2E latency than both state-based baselines at every measured context and output length. At 128K, it is 2.40–2.67× faster than δ-Mem and 6.46–8.71× faster than Temp-LoRA. Consistent with this gap, state writing contributes 71.7–90.8% of δ-Mem E2E latency at 128K, while temporary adaptation contributes 92.2–97.7% for Temp-LoRA.

Analysis. The E2E results reveal two complementary efficiency advantages. Relative to Full Context, Metis becomes increasingly advantageous as history grows in the long-context regime. The crossover is first observed only at 64K because Qwen3.5-4B already uses linear attention in 24 of its 32 layers, substantially reducing the context-dependent KV traffic of Full Context and thereby delaying the E2E crossover in favor of Metis. As the KV caches of the remaining 8 full-attention layers continue to grow, the advantage of Metis becomes more pronounced: when context doubles from 64K to 128K, Full Context prefill increases by approximately 2.67× (4.34 to 11.60 seconds) and decode by approximately 1.65× (1.40 to 2.31 seconds for 32-token outputs and 5.74 to 9.45 seconds for 128-token outputs), widening the E2E latency gap. Relative to the two state-based baselines, Metis maintains lower E2E latency throughout the entire context-length sweep because it avoids their increasingly expensive state writing or temporary adaptation. We use query latency only to isolate the cost of memory readout, estimated as Metis query latency minus No Context latency. For 32-token outputs, the estimated readout is 0.091 seconds at 64K and 0.106 seconds at 128K; for 128-token outputs, it is 0.770 and 0.559 seconds, respectively. These values account for only 7.2–14.4% of Metis query latency and do not increase when the history doubles. **Figure 8** similarly shows that pure memory commit contributes only 2.1–4.3% of Metis E2E latency at 64K and 128K, with most ingestion time spent in the history-encoding backbone forwards.

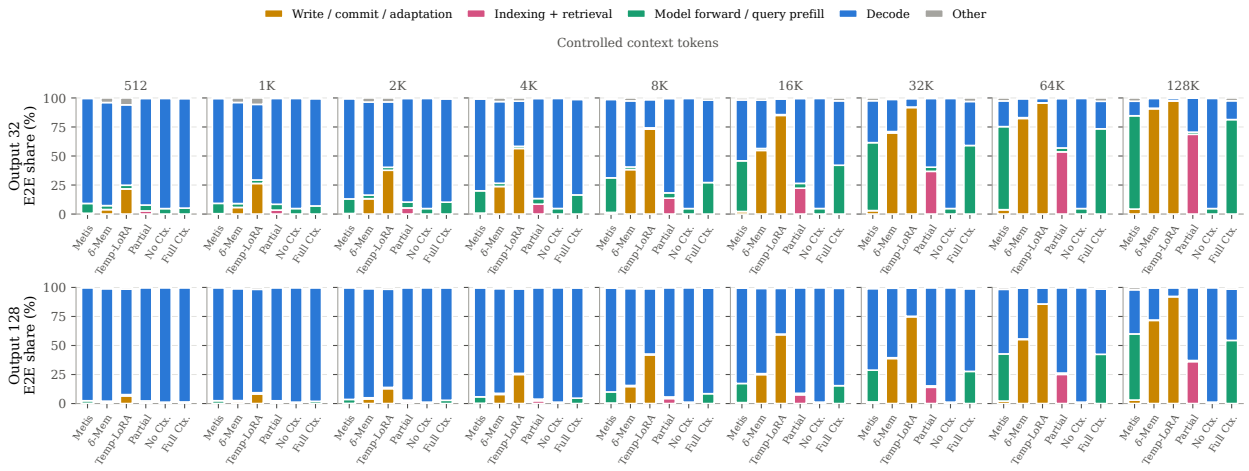

Figure 8 Normalized end-to-end latency breakdown across context lengths for 32-token (top) and 128-token (bottom) outputs. For Metis, model forward/query prefill includes the history-encoding forwards and final query prefill, while write/commit/adaptation contains only the pure memory update. For δ-Mem and Temp-LoRA, the latter category represents state writing and temporary adaptation, respectively; indexing and retrieval is reported separately for Partial Context. Other includes reset, tokenization, prompt construction, synchronization, text decoding, and unseparated host-side work.

Thus, neither memory readout nor pure memory commit is a major latency bottleneck. In principle, both stages can be executed concurrently with backbone attention on separate CUDA streams. Such asynchronous overlap, however, requires finer-grained kernel launches and inter-stream synchronization, introducing launch and scheduling overhead that can offset the latency hidden at this scale. We therefore do not enable this optimization in the final implementation.

G.3 Per-Session Storage across Context Lengths

Evaluation Protocol. We also measure the persistent per-session storage of the 4B methods over different context lengths from 512 to 32K token. All values are obtained from runtime objects, rather than estimated analytically from parameter counts. For Full Context, we retain the KV cache required to resume a session without replaying its history; the measurement includes the controlled context, a fixed 512-token query prompt, and 32 generated tokens. KV cache and model states are stored in BF16. The RAG store contains FP32 BGE-M3 embeddings and the corresponding chunk text. For state-based methods, we count only state that must be persisted separately for each session.

Results. Full Context storage grows from 87.06 MB at 512 context tokens to 1,118.86 MB at 32K. Its 8 full-attention layers add 32.8 KB of KV cache per token, while the 24 linear-attention layers contribute a constant 51.90 MB state. In contrast, Metis Full remains at 16.79 MB and Metis $k = 64$ remains at 2.11 MB. At 32K, Full Context therefore occupies 67× the Metis Full state and 529× the Metis $k = 64$ state; extrapolating the measured linear trend to 128K gives approximately 256× and 2,000×, respectively. The other constant-size states are 4.6 KB for δ-Mem and 190.3 MB for Temp-LoRA, whereas RAG Store grows from 0.104 MB at 512 tokens to 6.051 MB at 32K.

Analysis. Metis eliminates context-dependent storage growth: the Full Context KV cache reaches 1,066.96 MB at 32K, whereas both Metis states remain constant. Low-rank persistence provides a particularly efficient operating point: Metis ($k = 64$) uses one eighth of the Metis Full state while retaining 99.9% of full-state performance on average. δ-Mem occupies only 4.6 KB because its official rank-8 configuration stores one 8×8 BF16 online matrix in each of 36 attention modules, compressing each session into a low-dimensional state; this limited capacity is consistent with its lower LoCoMo score (Table 6). Despite the small state, every query token reads it in all 36 attention modules and applies additional $q/k/v/o$ delta projections, introducing per-layer matrix operations and kernel launches on top of the backbone attention. RAG Store also grows with history and depends on a separate retrieval and indexing pipeline; Metis $k = 64$ becomes smaller at approximately 11K tokens and remains smaller at the measured 16K and 32K points. Temp-LoRA requires a 190.3 MB per-session adapter, 11.3× the Metis Full state. Overall, Metis provides high storage efficiency for long texts: both Metis Full and $k = 64$ remain constant and substantially smaller than Full Context, while $k = 64$ also becomes smaller than RAG Store as history grows. As a storage-only upper bound on an 80 GB GPU with approximately 72 GB remaining after 4B model weights, 32K Full Context caches accommodate roughly 64 resident

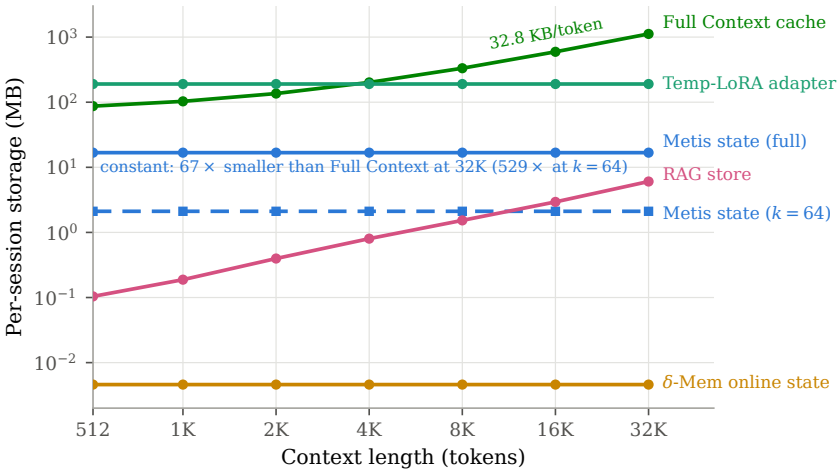

Figure 9 Persistent storage per session as a function of context length.

sessions, compared with about 4,000 full-state Metis sessions; the former decreases with history length, while the latter is independent of it.

G.4 Multi-User Serving

A critical concern for Metis is that maintaining a personalized memory for every user means deploying one model per user, which would be prohibitive at serving scale. In this section, we introduce a new serving solution for Metis to address this issue, which is inspired by LoRA-based approaches.

Serving Architecture. All users can share a single copy of the frozen backbone and the static Metis parameters. Therefore, the only per-user component is the dynamic memory state, which persists at 16.79 MB per user in full precision for Metis-4B. Based on this property, we refer to the design of multi-tenant LoRA serving systems such as Punica [87] and S-LoRA [88]. Specifically, we treat each user's dynamic memory state as a user-specific adapter, and design a **Memory-as-Adapter (MaA)** serving architecture for Metis that batches requests from different users into a single parallel forward pass. This architecture consists of three components. First of all, we consider states as adapters. Each user's memory state is persisted on SSD as swappable per-user data, gathered by user index when a session starts and written back upon eviction. The second component is stateless model replicas. A serving instance is bound to no user, so any replica can serve any request, making horizontal scaling and load balancing identical to ordinary LLM serving. Finally, we use batched state injection. When requests from different users are batched together, their states are stacked along the batch dimension and injected into the model in one operation. The key design is parallelism within the adapter layer itself. Generic multi-adapter inference has an inherent serialization bottleneck at the adapter computation. Since adapter shapes vary across tenants (*e.g.,* LoRA ranks), naive implementations must partition the batch by adapter and compute group by group, while efficient ones rely on customized gather-style kernels such as Punica. Metis eliminates this bottleneck by construction. All users' memory states share an identical shape, so the memory readout reduces to a single standard batched matrix multiplication. During this process, each request reads its own state within one kernel call, without grouping, sorting, or custom operators. Under MaA, from state injection to the end of decoding, the forward pass contains no per-user serial segment. The incremental cost of multi-user serving over single-user inference is merely a batch dimension.

Evaluation Protocol. In this subsection, we evaluate the performance of Metis during serving from three dimensions: isolation, layency scaling under batching, and the overhead of memory path. Each experiment includes one untimed warm-up, and CUDA synchronization is performed at all timing boundaries. All results are medians over five runs. The workload is a synthetic multi-user session. Each user holds an independent history containing user-specific facts, and each user's state is pre-built via standard write paths. The measurements of Metis account for the full overhead of loading and injecting state at request. Each baseline uses the most favorable accounting method. RAG pre-builds the vector database index, and latency only calculates retrieval and inference latency. Full Context is considered to have KV

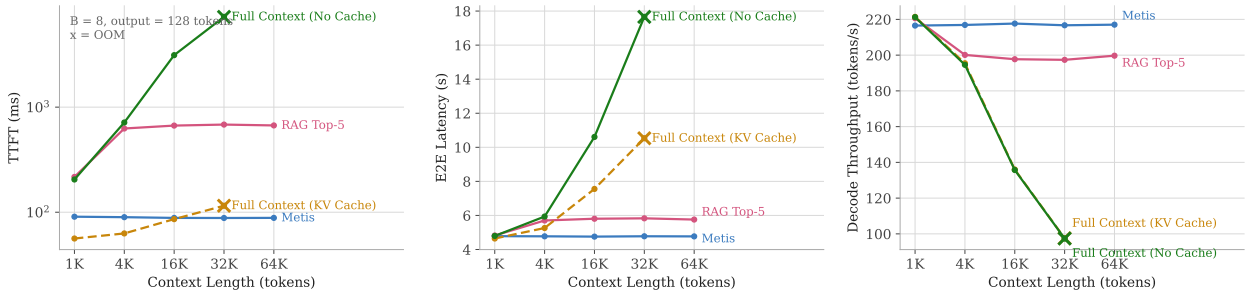

Figure 10 Batched multi-user serving (B=8, 128 generated tokens per user) as per-user context length grows from 1K to 64K tokens. Left: Time to First Token (TTFT). Middle: end-to-end latency per batched request. Right: decode-phase throughput.

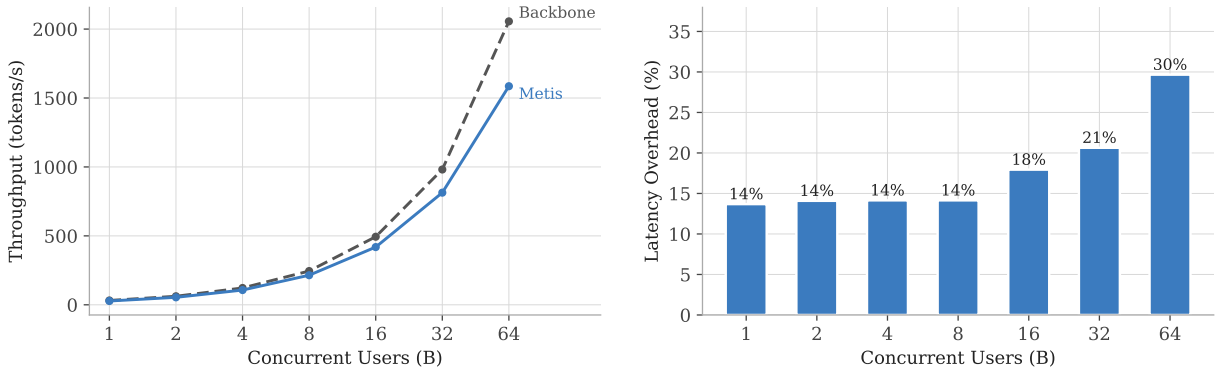

Figure 11 Serving cost of the native-memory path. Left: aggregate decode throughput of batched multi-user Metis. Right: relative latency overhead, which includes re-stacking and re-injecting all per-user states on every request.

Cache and no Cache, while the KV-cache variant assumes that the Context has a pre-built KV Cache residing in HBM.

Isolation. Because per-request states occupy disjoint slices of the batch dimension, cross-user interference is structurally excluded, and we verify this empirically with 20 concurrently served users holding globally unique persona facts. In evaluation, across 200 sampled cross-user probes, in which user i is asked about user j's fact, no response ever contained another user's value. We also test the recall of own-fact across various batch sizes, where we sweep $B \in \{1, 4, 8, 16, 20\}$ yields identical accuracy under a one-fact-per-user load (100.0% at every B).

Latency Scaling with Context Length under Batching. Figure 10 sweeps per-user context length from 1K to 64K tokens under the MaA architecture, with B=8 and 128 generated tokens per user, comparing Metis against RAG and Full Context with and without a resident KV cache in terms of TTFT, end-to-end latency, and decode throughput. All three Metis metrics are flat from 1K to 64K. TTFT stays at 88–91 ms (including loading and injecting states from SSD), end-to-end latency at 4.76–4.78 s, and decode throughput at 216–218 tokens/s.

In contrast, the Full Context (KV Cache) variant achieves lower TTFT at short contexts, but its decode throughput halves as context grows (221 to 97 tokens/s from 1K to 32K). Even with prefill fully amortized by the resident cache, every decoding step must attend over the entire 32K-token cache, whereas the Metis query processes a short prompt and reads a fixed-size state. Its end-to-end latency is therefore overtaken by Metis beyond 4K and reaches 2.2× the Metis latency at 32K. Without a cache, Full Context pays the full prefill on every request, and its TTFT grows from 0.2 to 7.2 s. At 64K, both Full Context variants run out of memory, as the batched request totals $B \times 64\text{K} \approx 525\text{K}$ prompt tokens, while Metis and RAG are unaffected. RAG plateaus at roughly 0.67 s TTFT and 5.8 s end-to-end, since retrieval bounds its prompt length; it is the closest baseline, but pays a fixed extra second per request and its per-session store grows unboundedly with history.

Overhead of the Memory Path. We further compare the latency of Metis-4B against the raw Qwen3.5-4B backbone under batching, to quantify the gap between Metis and the latency lower bound of the serving system. As shown

in **Figure 11**, under identical prompts and decoding settings, the full memory path adds only a stable 14% latency overhead up to B=8, rising to 30% at B=64. The latter is measured under a deliberately conservative protocol that re-loads, re-stacks, and re-injects all per-user states on every request. Despite this overhead, scaling remains near-linear. Aggregate throughput reaches 1,585.6 tokens/s at B=64, a 57.1× speedup over serving the same users serially, with only a 12% increase in per-user latency. This finding agrees with the single-request efficiency analysis above. As shown in **Section G.2**, at B=1 memory readout adds only about 0.1 s (7.2–14.4% of query latency), and pure commit accounts for merely 2.1–4.3% of end-to-end latency. In other words, the memory path is not the latency bottleneck, in either the single-request or the batched serving regime, and batching does not alter this property.

Analysis. In the above part, the MaA architecture provides a foundation for the large-scale deployment that Metis may face in the future. Under this architecture, the per-user cost of Metis is a fixed-size, swappable state rather than a model replica. Relative to context-replay baselines, its advantage compounds with context length and with the decode share of the workload. Compared with the raw backbone, the system overhead it introduces is limited. Moreover, since per-request states occupy disjoint slices of the batch dimension, the architecture excludes cross-request memory interference by construction. This simplifies data governance, because deleting a user reduces to deleting one state file.

We note that these results are single-node, static-batch prototype measurements reporting medians rather than tail latency under realistic load. Several systems challenges remain on the efficiency side. Integrating MaA with continuous batching requires online allocation and reclamation of state slots as requests dynamically join and leave a batch, In addition, memory writes are currently performed as offline commits, production serving will interleave reads (generation) with writes (memory updates), calling for state-consistency and scheduling support at the systems level. Nevertheless, the fixed size, uniform shape, and read-write separation of memory states provide a solid structural basis for these optimizations, and we believe large-scale deployment of Metis is well within reach.