MEMEDIT: FINE-GRAINED MEMORY SURGERY FOR EXPANDABLE PARAMETRIC AGENT MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Parametric memory modules, which are lightweight neural networks pretrained to internalize retrieval patterns, have emerged as a promising alternative to retrieval-augmented generation (RAG) for equipping large language model (LLM) agents with persistent knowledge. However, existing parametric memories operate as *write-only* stores: once knowledge is encoded into parameters, there is no principled mechanism for locating, updating, or erasing individual memory traces without costly retraining. This limitation is critical in long-lived agent scenarios where facts become outdated, user preferences shift, and privacy regulations mandate selective forgetting. We introduce **MemEdit**, a fine-grained *memory surgery* framework that brings the full spectrum of CRUD operations: **C**reate, **R**ead, **U**pdate, **D**elete, to parametric agent memory. MemEdit carries out every operation in four stages: *locate*, *preserve*, *intervene*, and *verify*. A *Memory Attribution* mechanism based on integrated gradients localizes each memory trace to a neuron footprint, which QUERY exposes for inspection. The three write operations share one constrained rank-one update whose output side is confined to the footprint and whose input side is projected away from the keys of preserved memories: (i) INSERT writes into the null space of existing keys with exact preservation; (ii) MODIFY rewrites a stored value under a locality penalty; and (iii) DELETE applies projected gradient ascent that leaves all other stored keys unchanged. Every write is journaled as a reversible low-rank delta and verified. When the null space is exhausted, a *Mixture-of-Memory-Experts* (MoME) expansion strategy adds independently editable memory shards, so that editing capacity grows with the number of shards. Experiments on LoCoMo and MemBench benchmarks show that MemEdit achieves 94.6% edit success rate while preserving 97.3% of unrelated memory accuracy, scales to over 10K sequential edits without catastrophic forgetting, and outperforms both non-parametric and parametric baselines in downstream QA.

1 INTRODUCTION

Memory is a cornerstone capability for autonomous LLM agents that operate over extended time horizons (Park et al., 2023; Wang et al., 2025; Yu et al., 2025). An agent tasked with personalized assistance, knowledge-intensive dialogue, or multi-session planning must remember past interactions, user preferences, and accumulated facts, all while remaining *efficient* at inference time. Two dominant paradigms have emerged for endowing LLMs with persistent memory. **Non-parametric** approaches store memories as explicit text, embeddings, or knowledge-graph nodes in an external database, retrieving relevant entries at each inference step via similarity search (Lewis et al., 2020; Packer et al., 2023; Xu et al., 2025; Kang et al., 2025). While interpretable and easy to update, these methods suffer from retrieval latency, top-K information truncation, and a fundamental decoupling between retrieval and generation that hinders end-to-end optimization.

Parametric approaches take a fundamentally different path: they compress knowledge into trainable parameters—an auxiliary MLP (Wei et al., 2025), a small decoder (Cao et al., 2025), or a set of prefix embeddings (Hu et al., 2022)—that are queried through a single forward pass at inference time, eliminating retrieval overhead entirely. MLP Memory (Wei et al., 2025) achieves $2.5\times$ faster inference than RAG with superior QA accuracy, and Memory Decoder (Cao et al., 2025) enables plug-and-play domain adaptation across heterogeneous LLM backbones. These advantages are most

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

pronounced for large or diffuse memories and for latency-constrained or on-device serving, while small, human-curated memories remain well served by explicit stores.

However, existing parametric memories are fundamentally *write-only*. Once knowledge has been encoded into the module's weights, whether through pretraining on kNN retrieval distributions or domain-specific distillation, there is no principled way to operate on individual memory entries. An outdated fact requires retraining the entire module, a deletion request under privacy regulations has no targeted erasure procedure, and conflicting information cannot be reconciled by locating and modifying the responsible parameters. Non-parametric systems, in contrast, naturally support CRUD (Create, Read, Update, Delete) operations at the entry level.

Meanwhile, the model editing community has developed sophisticated techniques for localizing and modifying factual associations within LLM weights. Causal tracing (Meng et al., 2022) identifies decisive MLP neurons, ROME and MEMIT (Meng et al., 2022; 2023) perform rank-one parameter updates to rewrite facts, and AlphaEdit (Fang et al.) projects perturbations onto null spaces to preserve unrelated knowledge. These methods are effective for LLM backbones but have not been applied to *external parametric memory modules*, which differ in architecture (all-MLP or small decoder), training objective (kNN distribution imitation rather than language modeling), and use case (agent-level memory management rather than static fact correction). They also treat each edit in isolation, with no notion of how much editing capacity remains after many edits.

In this paper, we bridge these two lines of work with **MemEdit**, a framework that carries out every memory operation as *memory surgery* in four stages: locate, preserve, intervene, and verify. The design rests on a simple observation: the footprint of a memory determines which output neurons of a weight matrix an update may touch, while the keys of stored memories determine which input directions it must avoid. Localization and preservation thus act on complementary sides of the same matrix, which lets all write operations share one constrained rank-one update and turns the remaining null-space dimension into a measurable editing capacity. Our contributions are as follows:

- We introduce a **Memory Attribution** mechanism that localizes memory traces with integrated gradients over the module's neurons, yielding a compact neuron footprint for each memory that the QUERY operation exposes for inspection.
- We derive a **unified write rule** whose output side is confined to the footprint and whose input side is projected away from the keys of preserved memories. INSERT, MODIFY, and DELETE are instances of this rule, with provable preservation of stored keys for Insert and Delete. Every write is journaled as a reversible low-rank delta and verified.
- We propose **Mixture-of-Memory-Experts (MoME)**, which monitors the null-space rank of each shard and adds a new, independently editable shard when editing capacity is exhausted, so that capacity grows linearly with the number of shards.
- Experiments on LoCoMo (Maharana et al., 2024), MemBench (Tan et al., 2025), and a 10K-edit stress test show that MemEdit achieves highest edit success rate and downstream QA accuracy among compared methods, while maintaining memory integrity under sequential editing at scale.

2 RELATED WORKS

Non-Parametric Agent Memory. Most LLM agents store experiences as text, embeddings, or knowledge-graph nodes in an external database and retrieve them by similarity search. Generative Agents (Park et al., 2023) pioneered reflection-augmented text memory for social simulation, and MemGPT (Packer et al., 2023) pages context in and out of the LLM's limited window in an operating-system analogy. A-Mem (Xu et al., 2025) adds agentic, Zettelkasten-inspired dynamic indexing and linking of memory notes. MemoryOS (Kang et al., 2025) organizes short-term, mid-term, and long-term personal memory with heat-driven retrieval, MIRIX (Wang & Chen, 2025) covers multimodal episodic, semantic, and procedural memories, and Mem0 (Chhikara et al., 2025) combines graph databases with RAG-based retrieval. These systems support entry-level CRUD naturally, but they pay retrieval latency at every step, lose context to top-K truncation, and decouple retrieval from generation, which prevents end-to-end gradient flow.

Parametric Memory. Parametric approaches compress knowledge into trainable parameters and remove retrieval overhead. MLP Memory (Wei et al., 2025) pretrains an all-MLP module to imitate

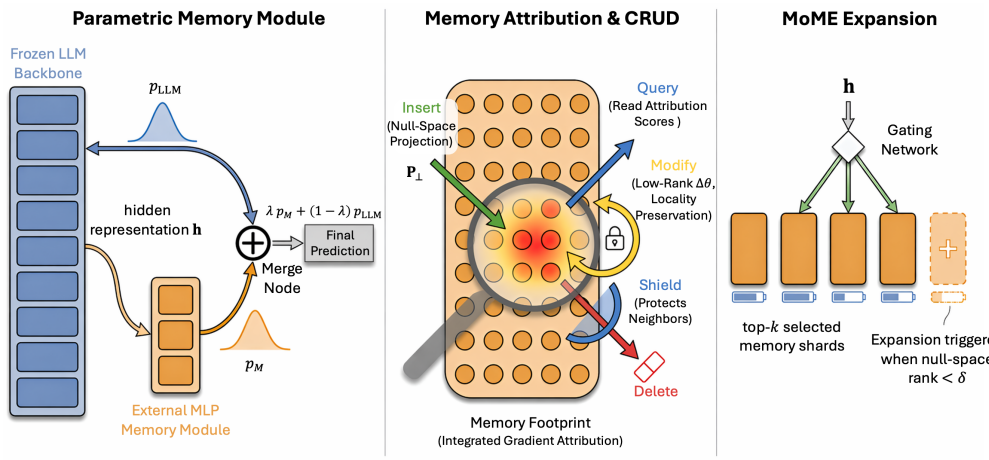

Figure 1: **Overview of MemEdit. Left:** The parametric memory module takes hidden representations from a frozen LLM and interpolates its output with the LLM's distribution. **Center:** Memory Attribution localizes memory traces to neuron subsets, enabling four CRUD operations. **Right:** MoME dynamically spawns new memory shards when null-space capacity is exhausted.

kNN retrieval distributions over the pretraining corpus and interpolates its output with the LLM's predictions, running 2.5× faster than RAG with higher accuracy. Memory Decoder (Cao et al., 2025) uses a small transformer decoder for plug-and-play domain adaptation, and LongMem (Wang et al., 2023) pairs a frozen backbone memory encoder with an adaptive side-network for retrieval and reading. LoRA (Hu et al., 2022) and prefix-tuning variants encode task- or user-specific adaptations into low-rank updates, and MemLoRA (Bini et al., 2025) distills expert adapters for on-device memory. None of them supports post-hoc editing of individual memory entries, the gap MemEdit addresses.

Knowledge Editing and Machine Unlearning. The "locate-then-edit" paradigm (Dai et al., 2022; Meng et al., 2022) identifies knowledge neurons by causal tracing and then applies targeted weight updates. ROME (Meng et al., 2022) performs rank-one modifications to MLP projection matrices, MEMIT (Meng et al., 2023) extends this to batch editing across layers, and AlphaEdit (Fang et al.) projects perturbations onto the null space of preserved knowledge. GRACE (Hartvigsen et al., 2023) maintains a dynamically updated codebook for continual editing, FiNE (Pan et al., 2025) localizes edits at the neuron level, and the unified framework of (Gupta et al., 2024) shows that ROME and MEMIT share a preservation-memorization objective differing only in constraint type. Machine unlearning (Jang et al., 2023) removes the influence of specific training samples via gradient ascent or parameter projection. All of these methods target the LLM backbone's own weights and treat edits in isolation. MemEdit transfers them to external parametric memory modules, whose single-vector inputs and kNN-imitation objective call for a different localization and preservation design, and adds an explicit notion of remaining editing capacity.

3 Methodology

MemEdit carries out every change to a parametric memory as a surgical procedure with four stages: *locate* the parameters that carry a memory (Section 3.2), *preserve* the memories that must remain intact and *intervene* with a minimal weight update (Section 3.3), and *verify* the outcome (Section 3.4).

3.1 Preliminaries: Parametric Memory Modules

We consider a parametric memory module $\mathcal{M}_\theta$ pretrained to internalize retrieval patterns from a knowledge corpus (Figure 1, left panel). Following (Wei et al., 2025) and (Cao et al., 2025), $\mathcal{M}_\theta$ takes the hidden representation $\mathbf{h} = f(c)$ of a context c from an intermediate layer of a frozen LLM and produces a token distribution $p_{\mathcal{M}}(y|c)$. During inference, this distribution is interpolated with the LLM's own output:

$$p(y|c) = \lambda \cdot p_{\mathcal{M}}(y|c) + (1-\lambda) \cdot p_{\text{LLM}}(y|c), \tag{1}$$

where $\lambda \in [0, 1]$ is a mixing coefficient. $\mathcal{M}_\theta$ is typically an all-MLP network (as in MLP Memory) or a small transformer decoder (as in Memory Decoder), pretrained with a combination of KL divergence to the kNN retrieval distribution and cross-entropy loss. We describe the all-MLP case, with L layers, weight matrices $\mathbf{W}_l^{(1)}, \mathbf{W}_l^{(2)}$ for the two linear projections of layer l, and neuron activations $\mathbf{a}_l$, so that $\theta = \{\mathbf{W}_l^{(1)}, \mathbf{W}_l^{(2)}\}_{l=1}^{L}$. A memory m_i (e.g., "the user prefers Italian cuisine") is represented by a probe input $\mathbf{h}_i$ and a target distribution p_i^*, both built from a declarative statement of m_i (Section 3.4). Its *memory trace* is the subset of parameters and activations causally responsible for the module's output on $\mathbf{h}_i$.

Key–value view. Following (Meng et al., 2022), we view the first projection of a target layer l^*, $\mathbf{W} \equiv \mathbf{W}_{l^*}^{(1)} \in \mathbb{R}^{d_{\text{ff}} \times d}$, as a linear key–value store. Its input $\mathbf{k} \in \mathbb{R}^d$ is the *key* of a probe, its output $\mathbf{W}\mathbf{k}$ is the *value*, and its rows correspond one-to-one to the neurons of layer l^*. We select l^* as the layer with the highest average edit success rate on a held-out validation set. MemEdit caches the key $\mathbf{k}_j$ and a sparse copy of p_j^* for every stored memory in a *key bank* $\mathbf{K}_0 = [\mathbf{k}_1, \ldots, \mathbf{k}_n]^\top \in \mathbb{R}^{n \times d}$. With $\mathbf{V}_r$ denoting the right singular vectors of $\mathbf{K}_0$ whose singular values exceed ϵ_{svd}, the projector onto the null space of $\mathbf{K}_0$ is

$$\mathbf{P}_\perp = \mathbf{I} - \mathbf{V}_r \mathbf{V}_r^\top. \tag{2}$$

It satisfies $\mathbf{P}_\perp \mathbf{k}_j = \mathbf{0}$ for every $\mathbf{k}_j$ in the row space of $\mathbf{K}_0$. We denote by $\mathbf{P}_\perp^{(i)}$ the projector computed from $\mathbf{K}_0^{(-i)}$, the key bank without $\mathbf{k}_i$.

3.2 Memory Attribution: Locating Memory Traces

For LLM backbones, causal tracing (Meng et al., 2022) corrupts subject tokens and restores individual hidden states to measure causal effects. Parametric memory modules, however, process a *single* vector without the token structure that causal tracing exploits. MemEdit therefore uses integrated gradients over the module's neurons. For a memory m_i, the attribution score of neuron j in layer l is

$$\mathcal{A}_{l,j}(m_i) = \left| \int_0^1 \frac{\partial \mathcal{L}\big(\mathcal{M}_\theta(\tilde{\mathbf{h}}_i(\alpha)), p_i^*\big)}{\partial a_{l,j}} \cdot \frac{\partial a_{l,j}\big(\tilde{\mathbf{h}}_i(\alpha)\big)}{\partial \alpha} \, d\alpha \right|, \tag{3}$$

where $\tilde{\mathbf{h}}_i(\alpha) = \alpha \mathbf{h}_i + (1 - \alpha)\bar{\mathbf{h}}$ interpolates between a baseline $\bar{\mathbf{h}}$ (the mean representation over the pretraining corpus) and the probe, $a_{l,j}$ is the activation of the neuron, and $\mathcal{L}$ is the KL divergence. The integral is approximated with $S = 20$ Riemann steps. The *memory footprint* of m_i at sparsity τ is the top-τ fraction of neurons across all layers:

$$\mathcal{F}_\tau(m_i) = \{(l, j) \mid \mathcal{A}_{l,j}(m_i) > \text{Quantile}_{1-\tau}(\{\mathcal{A}_{l',j'}(m_i)\}_{l',j'})\}. \tag{4}$$

Restricted to layer l^*, the footprint defines a diagonal row mask $\mathbf{M}_i$ with $[\mathbf{M}_i]_{jj} = \mathbb{I}[(l^*, j) \in \mathcal{F}_\tau(m_i)]$, which confines later edits of m_i to the rows of $\mathbf{W}$ that carry it.

Query. Query exposes localization as the read operation of the CRUD interface. Given a named memory m_i, it returns $\mathcal{F}_\tau(m_i)$ and a confidence c_i, defined as the share of the total attribution score carried by the footprint. Query is not needed to answer user turns, since reading memory is the single forward pass of Eq. (1). It is invoked by write operations and lets agents inspect which parameter regions encode which memories, a form of parametric interpretability.

3.3 Memory Writes as Constrained Rank-One Surgery

All three write operations modify only $\mathbf{W}$, and each update takes the form

$$\Delta\mathbf{W} = \mathbf{M}\,\mathbf{u}\,\mathbf{q}^\top, \qquad \text{so that} \qquad \Delta\mathbf{W}\mathbf{k} = (\mathbf{q}^\top \mathbf{k})\,\mathbf{M}\mathbf{u} \;\text{ for any key } \mathbf{k}. \tag{5}$$

Here $\mathbf{u} \in \mathbb{R}^{d_{\text{ff}}}$ carries the change in value, the diagonal mask $\mathbf{M}$ selects the rows that may change, and $\mathbf{q} \in \mathbb{R}^d$ determines which keys are affected. Localization acts on the output side through $\mathbf{M}$, and preservation acts on the input side: every key with $\mathbf{q}^\top \mathbf{k} = 0$ keeps its value exactly. The operations differ only in how they set $\mathbf{u}$, $\mathbf{q}$, and $\mathbf{M}$ (Figure 1, center panel).

Insert. A new memory with key $\mathbf{k}_{\text{new}}$ must be written without altering any stored value. As in ROME (Meng et al., 2022), we first compute a target value $\mathbf{v}_{\text{new}}$ by optimizing the output of layer l^* so that the module produces p_{new}^*. Inspired by the null-space projection of AlphaEdit (Fang et al.),

we impose preservation on the input side by setting $\mathbf{q} \propto \mathbf{P}_\perp \mathbf{k}_{\text{new}}$, and we use $\mathbf{M} = \mathbf{I}$ since a new memory has no footprint yet:

$$\Delta \mathbf{W}_{\text{ins}} = \frac{(\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}})(\mathbf{P}_\perp \mathbf{k}_{\text{new}})^\top}{\|\mathbf{P}_\perp \mathbf{k}_{\text{new}}\|^2}. \tag{6}$$

Proposition 1 (Insert Invariance and Exactness)**.** *Let $\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}_{ins}$ as in Eq.* (6)*, with $\mathbf{P}_\perp \mathbf{k}_{new} \neq \mathbf{0}$. Then (i) $\mathbf{W}'\mathbf{k}_j = \mathbf{W}\mathbf{k}_j$ for every $\mathbf{k}_j$ in the row space of $\mathbf{K}_0$, and (ii) $\mathbf{W}'\mathbf{k}_{new} = \mathbf{v}_{new}$.*

Both parts follow from $\mathbf{P}_\perp$ being symmetric and idempotent: $(\mathbf{P}_\perp \mathbf{k}_{\text{new}})^\top \mathbf{k}_j = \mathbf{k}_{\text{new}}^\top \mathbf{P}_\perp \mathbf{k}_j = 0$ and $(\mathbf{P}_\perp \mathbf{k}_{\text{new}})^\top \mathbf{k}_{\text{new}} = \|\mathbf{P}_\perp \mathbf{k}_{\text{new}}\|^2$ (Appendix A). If $\mathbf{k}_{\text{new}}$ is nearly spanned by stored keys, i.e., $\|\mathbf{P}_\perp \mathbf{k}_{\text{new}}\|/\|\mathbf{k}_{\text{new}}\| < \delta_{\text{key}} = 0.05$, the insert is redirected to a fresh shard (Section 3.5); otherwise, $\mathbf{k}_{\text{new}}$ is appended to $\mathbf{K}_0$.

Modify. Modify updates a memory m_i to new content m_i' (e.g., from "user lives in Tokyo" to "user lives in Osaka"), keeping its key $\mathbf{k}_i$ while changing its value. After Query locates $\mathcal{F}_\tau(m_i)$, we solve

$$\min_{\Delta\theta} \; \mathcal{L}(\mathcal{M}_{\theta+\Delta\theta}(\mathbf{h}_i), p_i'^*) + \gamma \sum_{j \in \mathcal{U}} \|\mathcal{M}_{\theta+\Delta\theta}(\mathbf{h}_j) - \mathcal{M}_\theta(\mathbf{h}_j)\|_2^2, \tag{7}$$

where $p_i'^*$ is the target distribution of m_i', $\mathcal{U}$ is a representative set of unrelated memories drawn from the key bank, and $\gamma > 0$ controls the locality-preservation strength. The update is restricted to the footprint rows of $\mathbf{W}$. Following MEMIT (Meng et al., 2023), we compute a target value $\mathbf{v}_i'$ for $p_i'^*$ as in Insert and solve Eq. (7) as a least-squares problem at layer l^*. Because the edit concerns the single key $\mathbf{k}_i$, its closed-form solution is again an instance of Eq. (5), with $\mathbf{M} = \mathbf{M}_i$, $\mathbf{u} = \mathbf{v}_i' - \mathbf{W}\mathbf{k}_i$, and an input direction $\mathbf{q}$ that suppresses the response to the keys in $\mathcal{U}$. Insert and Delete impose preservation as a hard constraint over the key bank, whereas Modify imposes it as a penalty over $\mathcal{U}$.

Delete. Erasing m_i is the most challenging operation, since zeroing its footprint neurons would damage overlapping traces. MemEdit instead applies gradient-ascent unlearning (Jang et al., 2023) and protects all other stored keys through $\mathbf{P}_\perp^{(i)}$. Each step updates

$$\mathbf{W} \leftarrow \mathbf{W} + \eta_t \, \mathbf{M}_i \, \mathbf{G}_t \, \mathbf{P}_\perp^{(i)}, \qquad \mathbf{G}_t = \nabla_{\mathbf{W}} \mathcal{L}\big(\mathcal{M}_\theta(\mathbf{h}_i), p_i^*\big), \tag{8}$$

with a step size η_t that decays over T steps. For a single probe, $\mathbf{G}_t = \boldsymbol{\delta}_t \mathbf{k}_i^\top$, where $\boldsymbol{\delta}_t$ is the loss gradient with respect to the layer output. Each step is thus an instance of Eq. (5) with $\mathbf{M} = \mathbf{M}_i$, $\mathbf{u} = \eta_t \boldsymbol{\delta}_t$, and $\mathbf{q} = \mathbf{P}_\perp^{(i)} \mathbf{k}_i$. The ascent pushes the module away from recalling m_i, while the projection keeps all other stored values fixed.

Proposition 2 (Delete Soundness)**.** *After any number of steps of Eq.* (8)*, $\mathbf{W}\mathbf{k}_j$ is unchanged for every $\mathbf{k}_j$ in the row space of $\mathbf{K}_0^{(-i)}$. For the all-MLP module, the outputs $\mathcal{M}_\theta(\mathbf{h}_j)$ of the corresponding memories are therefore unchanged.*

The proof applies the argument of Proposition 1 to each step (Appendix A). Propositions 1 and 2 cover every memory whose key is in the key bank. Inputs outside it are protected only insofar as they lie close to the span of stored keys, which we measure in Appendix C.6.

3.4 Memory Lifecycle and Verification

Memory lifecycle. Writes are issued by an operation controller: the LLM backbone is prompted to map each dialogue turn to Insert, Modify, Delete, or None, together with a self-contained memory statement (Appendix D). The statement is split into a prefix, which yields the probe $\mathbf{h}_i$, and a target span, which yields p_i^* (Appendix B.2). Modify and Delete first run Query on the named memory to obtain its footprint. If $c_i < 0.2$, the memory is treated as absent: a Modify becomes an Insert, and a Delete is skipped. Insert records the footprint of the new memory after writing. Algorithm 1 summarizes one operation.

Verification and journaling. Each write ends with a verification probe that costs one forward pass. Insert and Modify must reproduce their target, Delete must raise $\mathrm{KL}(p_i^* \,\|\, \mathcal{M}_{\theta'}(\mathbf{h}_i))$ above a threshold τ_{del}, and the 20 nearest stored memories in the key bank must keep their outputs. Since every write is rank-one, or a sum of at most T rank-one steps for Delete, MemEdit journals it compactly as its low-rank factors with a typed, human-readable record (Appendix B.7). A failed check triggers rollback, $\theta \leftarrow \theta' - \Delta\theta$, and re-dispatch. A deletion is held as a tombstone with its delta retained until it is confirmed, so it stays reversible until committed.

Algorithm 1 MemEdit: One Memory Operation

Require: Memory module $\mathcal{M}_\theta$ with key bank $\mathbf{K}_0$; operation $\texttt{op} \in \{\textsc{Insert}, \textsc{Modify}, \textsc{Delete}\}$; memory statements m_i, m_i', or m_{new}.

Build probe $\mathbf{h} = f(c)$, key $\mathbf{k}$, and target p^* from the statement (Appendix B.2)
if $\texttt{op} \in \{\textsc{Modify}, \textsc{Delete}\}$ **then**
  QUERY: compute $\mathcal{A}_{l,j}(m_i)$, $\mathcal{F}_\tau(m_i)$, $\mathbf{M}_i$, and c_i via Eqs. (3)–(4)
  if $c_i < 0.2$ **then**
    if $\texttt{op} = \textsc{Delete}$ **then return** θ; **else** set $\texttt{op} \leftarrow \textsc{Insert}$ and $m_{\text{new}} \leftarrow m_i'$
  end if
end if
if $\texttt{op} = \textsc{Insert}$ **then**
  if $\|\mathbf{P}_\perp \mathbf{k}_{\text{new}}\| < \delta_{\text{key}} \|\mathbf{k}_{\text{new}}\|$ **then** redirect to a fresh shard (Section 3.5)
  Apply the projected rank-one update of Eq. (6) and append $\mathbf{k}_{\text{new}}$ to $\mathbf{K}_0$
else if $\texttt{op} = \textsc{Modify}$ **then**
  Solve Eq. (7) in closed form on the footprint rows selected by $\mathbf{M}_i$
else if $\texttt{op} = \textsc{Delete}$ **then**
  for $t = 1, \ldots, T$ **do**
    Apply the projected gradient-ascent step of Eq. (8)
  end for
end if
Verify the edit; roll back and re-dispatch on failure, otherwise journal the low-rank delta
return Updated parameters θ' and key bank $\mathbf{K}_0$

3.5 MIXTURE-OF-MEMORY-EXPERTS (MOME) EXPANSION

Each insertion consumes one dimension of the null space of $\mathbf{K}_0$. Because this editing capacity is directly observable as $\text{rank}(\mathbf{P}_\perp)$, MemEdit expands the memory when it runs out, using a Mixture-of-Memory-Experts (MoME) architecture (Figure 1, right panel) inspired by sparsely-gated mixtures of experts (Jacobs et al., 1991; Shazeer et al., 2017).

MoME maintains a collection of N memory shards $\{\mathcal{M}_{\theta_1}, \ldots, \mathcal{M}_{\theta_N}\}$, each an independently editable parametric memory module with its own key bank. A lightweight gating network G_ϕ routes each input to the top-k most relevant shards:

$$G_\phi(\mathbf{h}) = \text{TopK}\left(\text{Softmax}(\mathbf{W}_g \mathbf{h} + \mathbf{b}_g), k\right), \tag{9}$$

and the final memory output is:

$$p_{\text{MoME}}(y|c) = \sum_{n \in \text{TopK}} G_\phi(\mathbf{h})_n \cdot p_{\mathcal{M}_{\theta_n}}(y|c). \tag{10}$$

When the null-space rank of the active shard drops below a threshold, i.e., $\text{rank}(\mathbf{P}_\perp) < \delta \cdot d$, a new shard $\mathcal{M}_{\theta_{N+1}}$ is initialized by cloning the pretrained base memory weights and added to the mixture. Inserts that fail the δ_{key} test of Section 3.3 are redirected to a fresh shard in the same way. The gate gains one output unit and is fine-tuned on a replay buffer of 1,000 recent routing examples, each a pooled key with its shard label (Appendix B.5). Each CRUD operation targets only the shard selected by G_ϕ, so edits leave the parameters of all other shards unchanged.

Proposition 3 (Expansion Scalability)**.** *Under the MoME architecture, the total editable memory capacity scales as $\mathcal{O}(N \cdot d_{null})$, where N is the number of shards and d_{null} is the initial null-space dimension of each shard. Sequential edits within a single shard reduce its d_{null}; expansion to a new shard fully restores the available capacity.*

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

Datasets We use two public benchmarks and one constructed stress test. **(1) LoCoMo** (Maharana et al., 2024) contains 10 extended multi-session dialogues, each spanning up to 32 sessions with an average of 600 turns and 16K tokens, annotated with single-hop, multi-hop, temporal-reasoning,

Table 1: Main results on LoCoMo (F1) and MemBench tasks. ESR, MP, and FR are reported on MemBench. Results are mean ± std. Among parametric methods, including the adapted editing baselines, the best results are in **bold** and the second best are underlined.

Method	ESR (%) ↑	MP (%) ↑	FR (%) ↑	MemBench Acc. ↑	LoCoMo F1 ↑
Non-parametric baselines					
RAG-Base	91.2±0.8	99.1±0.2	95.3±0.6	62.1±1.2	34.7±0.9
A-Mem	93.5±0.6	98.7±0.3	96.1±0.5	67.4±0.9	38.2±0.7
MemoryOS	92.8±0.7	98.9±0.2	94.7±0.8	69.8±1.0	42.5±1.1
Mem0	90.4±1.0	99.0±0.3	93.8±0.9	64.5±1.1	36.1±1.3
Parametric baselines					
MLP Memory	—	—	—	71.2±0.7	44.8±0.8
Memory Decoder	—	—	—	70.5±0.8	43.1±0.9
LoRA-FT	82.3±1.4	78.5±2.1	71.2±2.8	63.8±1.6	39.6±1.5
DAPT	79.1±1.8	74.2±2.5	68.5±3.1	60.1±1.9	37.2±1.7
Editing baselines (adapted)					
ROME-M	88.7±1.1	89.3±1.3	82.1±1.7	68.9±1.1	41.3±1.0
MEMIT-M	90.2±0.9	91.5±1.1	84.6±1.5	70.3±0.9	42.7±0.8
AlphaEdit-M	92.4±0.7	94.8±0.6	88.3±1.2	72.6±0.8	44.1±0.7
MemEdit (Ours)	**94.6**±0.5	**97.3**±0.3	**93.1**±0.7	**75.8**±0.6	**46.2**±0.6

event-summarization, and adversarial (unanswerable) questions. We use its QA split to evaluate memory accuracy before and after editing. **(2) MemBench** (Tan et al., 2025) evaluates agent memory across factual knowledge retention, temporal knowledge updating, and selective forgetting. We build edit sequences from its multi-session dialogues, treating each knowledge update as a Modify operation and deriving Insert and Delete tasks from its factual memory entries. **(3) MemEdit-Stress** is a new test set of 10K sequential operations (2.5K each for Insert, Modify, Delete, and mixed operations) derived from CounterFact (Meng et al., 2022) and adapted to the memory-module setting to evaluate scalability. MemBench and MemEdit-Stress specify operation types and memory contents, while on LoCoMo the LLM backbone acts as the operation controller of Section 3.4 (Appendix D).

Baselines We compare against eleven baselines: *Non-parametric:* **(1) RAG-Base**, a FAISS vector store with BM25 hybrid retrieval; **(2) A-Mem** (Xu et al., 2025); **(3) MemoryOS** (Kang et al., 2025); and **(4) Mem0** (Chhikara et al., 2025). *Parametric:* **(5) MLP Memory** (Wei et al., 2025); **(6) Memory Decoder** (Cao et al., 2025); **(7) LoRA-FT**, LoRA fine-tuning (Hu et al., 2022) of the memory module on each edit batch; and **(8) DAPT**, domain-adaptive pretraining (Gururangan et al., 2020) continued on updated knowledge. *Editing methods adapted to the memory module:* **(9) ROME-M** (Meng et al., 2022), **(10) MEMIT-M** (Meng et al., 2023) for batch editing, and **(11) AlphaEdit-M** (Fang et al.) with null-space editing.

Evaluation Metrics **(1) Edit Success Rate (ESR)**: the fraction of edit operations whose output on the edited probe matches the intended target, or, for deletion, falls below a confidence threshold. **(2) Memory Preservation (MP)**: the fraction of unedited memories whose outputs stay within $\epsilon = 0.01$ KL divergence of their pre-edit values. **(3) Forgetting Ratio (FR)**: the fraction of deleted memories whose output probability falls below the random baseline. **(4) QA Accuracy**: answer F1 on LoCoMo and multiple-choice accuracy on MemBench, measuring end-to-end downstream impact.

Implementation Details We implement MemEdit on three backbones: **Qwen3-8B** and **Qwen3-14B** (Yang et al., 2025), and **Llama-3.1-8B** (Grattafiori et al., 2024). The memory module is a 4-layer MLP with hidden dimension 4096 and GeLU activations, pretrained on WikiText-103's kNN distributions extracted at 75% backbone depth. Attribution uses $S = 20$ Riemann steps with the corpus-mean baseline and footprint sparsity $\tau = 0.03$. Modify uses $\gamma = 0.1$, and Delete uses $T = 5$ gradient-ascent steps with initial $\eta = 0.01$ and cosine decay. MoME starts with $N = 2$ shards, expansion threshold $\delta = 0.15$, and top-$k = 1$ routing through a single linear gate followed by softmax. All experiments run on 8×A100 80GB GPUs. Unless otherwise stated, we report results on **Qwen3-8B** with single-pass edits (no rollback), as means and stds over 3 runs with different seeds.

4.2 Results and Analysis

Main Results Table 1 presents the main results. First, non-parametric baselines achieve high ESR, MP, and FR by design, since editing a database entry is trivial, but they lag on downstream QA due

Table 2: ESR (%) and MP (%) under sequential editing on MemEdit-Stress (Qwen3-8B backbone). "†" denotes model collapse (perplexity $> 10^4$). Results are mean over 3 runs.

	100 edits		500 edits		1K edits		5K edits		10K edits	
Method	ESR	MP	ESR	MP	ESR	MP	ESR	MP	ESR	MP
LoRA-FT	82.3	78.5	74.1	63.2	65.7	51.8	†	†	†	†
ROME-M	88.7	89.3	83.2	81.5	76.4	72.1	58.3	54.7	†	†
MEMIT-M	90.2	91.5	86.8	86.3	81.5	79.4	67.2	63.8	52.1	48.6
AlphaEdit-M	92.4	94.8	90.1	92.3	87.3	89.1	78.5	81.2	71.3	74.8
MemEdit (single)	94.6	97.3	92.8	95.7	90.1	93.2	82.6	86.4	75.8	79.3
MemEdit + MoME	**94.6**	**97.3**	**93.5**	**96.4**	**92.3**	**95.8**	**90.7**	**94.1**	**89.2**	**93.5**

Table 3: Per-operation breakdown on MemBench (100 operations per type). Latency is wall-clock time per operation on a single A100.

Operation	**Success Rate (%)**	**Preservation (%)**	**Latency (ms)**	**Downstream ΔF1**
Query	97.2±0.4	100.0	12.3±0.8	0.0
Insert	95.1±0.6	98.4±0.3	34.7±2.1	+1.8±0.3
Modify	93.8±0.8	96.7±0.5	41.2±2.8	−0.3±0.2
Delete	91.4±1.0	96.1±0.6	52.8±3.5	−0.6±0.3

to retrieval limitations; A-Mem and MemoryOS improve over RAG-Base through richer memory organization. Second, naïve parametric editing via LoRA-FT and DAPT achieves moderate ESR but severely degrades MP, confirming that unregulated parameter updates cause catastrophic interference. Third, adapted model-editing methods perform substantially better, with AlphaEdit-M's null-space projection providing the best baseline at 92.4% ESR and 94.8% MP. Fourth, MemEdit outperforms all parametric and editing baselines on every metric, with 94.6% ESR (+2.2% over AlphaEdit-M) and 97.3% MP (+2.5%). It also achieves the highest ESR and downstream QA performance among all methods, showing that attribution-guided, null-space-constrained surgery is well suited to the memory module's structure.

Scalability under Sequential Editing Table 2 highlights the role of MoME. All single-module baselines degrade as edits accumulate: LoRA-FT collapses beyond 1K edits, ROME-M beyond 5K, and even AlphaEdit-M falls to 71.3% ESR at 10K edits. Single-shard MemEdit declines more slowly but follows the same trend. With MoME enabled, MemEdit maintains 89.2% ESR and 93.5% MP at 10K edits, adding three shards (five in total) along the way. Figure 2 shows the resulting "staircase recovery" pattern: ESR dips slightly as the active shard fills up and recovers after each expansion (red triangles at about 1.2K, 3.8K, and 7.5K edits), as each new shard restores null-space capacity. Early memories are retained as well: recall of the first 100 inserted memories remains 94.3% after 10K edits, compared with 71.9% for AlphaEdit-M (Appendix C.8).

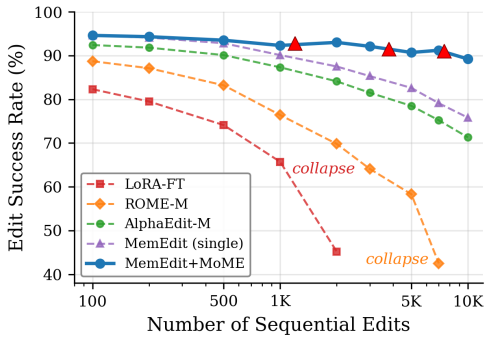

Figure 2: ESR (%) vs. number of sequential edits. MemEdit+MoME maintains high edit success by dynamically expanding memory shards (indicated by ▲).

Per-Operation Analysis Table 3 decomposes MemEdit's performance by operation type. Read-only Query achieves the highest success rate (97.2%) without affecting preserved memories. Insert follows at 95.1% success with 98.4% preservation, benefiting from its exact null-space guarantee. Modify and Delete are progressively harder, and Delete shows the lowest success rate (91.4%) because a distributed memory trace is difficult to erase fully without touching overlapping information. Under paraphrased and adversarial probes, FR drops to 84.7% and 76.5%, still above gradient ascent alone and AlphaEdit-M (Appendix C.7). Insertions improve downstream QA (+1.8 F1), whereas deletions

Table 4: Ablation study on key components (MemBench, 500 mixed edits, Qwen3-8B backbone).

Variant	ESR (%)	MP (%)
MemEdit (full)	**94.6**	**97.3**
w/o Memory Attribution (random localization)	81.3	84.7
w/o null-space projection (Insert)	93.1	88.5
w/o null-space projection (Delete)	90.8	86.2
w/o locality constraint (Modify)	92.7	89.1
w/o MoME (single shard)	93.9	95.8
$\tau = 0.01$	93.8	97.5
$\tau = 0.03$ (default)	94.6	97.3
$\tau = 0.05$	94.2	96.1
$\tau = 0.10$	93.5	93.8

cause a small decrease (-0.6 F1), as expected when removing information. All operations complete in under 60ms, which suits real-time agents.

Ablation Studies Table 4 validates each component of MemEdit. Replacing Memory Attribution with random neuron selection causes the largest degradation (-13.3% ESR, -12.6% MP), confirming that precise localization is the foundation of effective editing. Removing null-space projection from Insert maintains ESR, since the write still reaches its target, but drops MP by 8.8% through interference with existing memories. The locality constraint in Modify plays a similar role (-8.2% MP without it). For the sparsity threshold, a small $\tau = 0.01$ misses some relevant neurons and slightly reduces ESR, while a large $\tau = 0.10$ includes irrelevant neurons and reduces both ESR and MP; $\tau = 0.03$ achieves the best balance. Assembling the same primitives off the shelf reaches only 88.9% ESR at 500 edits and 69.2% at 10K edits on MemEdit-Stress; adding MemEdit's design choices one at a time recovers 4.6 and 20.0 points (Appendix C.2).

Cross-Backbone Generalization Table 5 shows that MemEdit generalizes well across LLM backbones. On the larger Qwen3-14B, MemEdit achieves slightly better results (95.2% ESR, 97.8% MP), suggesting that higher-quality backbone representations yield more separable memory traces. On Llama-3.1-8B, it achieves 93.9% ESR and 96.7% MP. The attribution and editing operations are backbone-agnostic because they act on the memory module's own parameters, whose architecture is the same for every backbone. The gap between MemEdit and AlphaEdit-M also remains consistent across the three backbones, indicating that the gains stem from memory-specific attribution and locality constraints rather than backbone-specific tuning.

Table 5: Evaluation results on different Qwen and Llama backbones (MemBench, 200 mixed edits).

	Qwen3-8B		Qwen3-14B		Llama-3.1-8B	
Method	ESR	MP	ESR	MP	ESR	MP
ROME-M	88.7	89.3	89.5	90.1	87.2	88.4
AlphaEdit-M	92.4	94.8	93.1	95.5	91.4	93.9
MemEdit	**94.6**	**97.3**	**95.2**	**97.8**	**93.9**	**96.7**

5 Conclusions and Future Work

We presented MemEdit, the first framework that enables fine-grained CRUD operations on parametric agent memory. MemEdit localizes memories through attribution, writes through a unified constrained rank-one rule that separates localization on the output side from preservation on the input side, verifies and journals every edit, and expands through Mixture-of-Memory-Experts when editing capacity runs out. This design scales to tens of thousands of sequential edits without catastrophic forgetting, with real-time latency suitable for agent deployment. Experiments on LoCoMo and MemBench demonstrate consistent improvements over parametric and editing baselines in edit fidelity and memory preservation, and over all baselines in downstream task performance. Future directions include extending MemEdit to multimodal parametric memories that store visual and auditory experiences, co-optimizing the memory module and its editing capacity through continual pretraining, investigating the interplay between MoME routing quality and memory retrieval accuracy at very large shard counts (e.g., > 100), and developing formal privacy guarantees for the Delete operation to comply with data protection regulations such as GDPR by European Union.

AI USE STATEMENT

In this work, we used generative AI tools only to improve the readability of the manuscript. Specifically, we used an LLM-based writing assistant to correct grammatical errors, fix typographical mistakes, and polish the phrasing of sentences that had already been drafted by the authors. We did not use generative AI tools for any of the tasks on the required-disclosure list. The research ideas and hypotheses, the conceptual framework, the mathematical claims and their proofs, the design of the methodology and the experiments, the implementation of all methods, the construction and processing of the datasets, and the interpretation of the results were produced entirely by the authors. All AI-assisted edits were reviewed by at least two authors to confirm that the original technical meaning was preserved and that no unsupported claims, citations, or factual statements were introduced during polishing. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

We have read and adhere to the ICLR Code of Ethics. This work does not involve human subjects, and all experiments are conducted on publicly available benchmarks (LoCoMo, MemBench, LongMemEval, CounterFact, WikiText-103) and publicly released model checkpoints (Qwen3-8B, Qwen3-14B, Llama-3.1-8B), used in accordance with their respective licenses and intended research purposes. The MemEdit-Stress test set is derived from CounterFact and contains no personally identifiable or sensitive information. The illustrative memory entries reported in our qualitative examples are synthetic and do not correspond to any real individual. We note two ethical considerations specific to our contribution. First, the Delete operation is motivated in part by data protection requirements such as the right to be forgotten, but as discussed in Appendix E it provides only empirical erasure rather than a formal unlearning or privacy guarantee, and residual traces of a deleted memory may persist in the parameter space. Practitioners should therefore not treat MemEdit as a certified compliance mechanism for regulations such as GDPR without further verification. Second, the same machinery that allows outdated facts to be corrected could in principle be used to insert false or misleading information into an agent's memory, or to selectively suppress accurate information. This risk is shared by the broader model editing literature, and we believe the benefits of giving practitioners auditable control over what an agent remembers outweigh it, provided that editing operations in deployed systems are logged and access-controlled. We declare no conflicts of interest or sponsorship that could have influenced the design or reporting of this work.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure that the results reported in this paper can be reproduced. The full specification of the parametric memory module, including its architecture, pretraining corpus, loss formulation, and all pretraining hyperparameters, is given in Appendix B, while Appendices B.2 through B.7 document the remaining implementation details of our method, covering probe and target construction, the integrated-gradient attribution procedure, the key bank and the truncated-SVD computation of the null-space projection, the MoME gating network and its expansion procedure, the schedule used for the Delete operation, and the verification and journaling protocol. The editing hyperparameters used in the main experiments, together with the backbones, hardware, and the number of random seeds, are reported in the Implementation Details paragraph of Section 4.1, and Section 4.2 and Appendix C report ablations, sensitivity analyses over the main hyperparameter choices, and the timing protocol. Complete proofs of Propositions 1, 2, and 3, including the assumptions each result relies on and a discussion of where the layer-wise guarantee holds only approximately, are provided in Appendix A. All benchmarks used in our evaluation are publicly available, and the construction of the MemEdit-Stress test set from CounterFact, along with the prompt used to determine edit operations on LoCoMo and LongMemEval, is described in Section 4.1 and Appendix D respectively. Anonymized source code for MemEdit is available at `https://anonymous.4open.science/r/MemEdit-4646`.

REFERENCES

Massimo Bini, Ondrej Bohdal, Umberto Michieli, Zeynep Akata, Mete Ozay, and Taha Ceritli. Memlora: Distilling expert adapters for on-device memory systems. *arXiv preprint arXiv:2512.04763*, 2025.

Jiaqi Cao, Jiarui Wang, Rubin Wei, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Memory decoder: A pretrained, plug-and-play memory for large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. Alphaedit: Null-space constrained knowledge editing for language models. In *The Thirteenth International Conference on Learning Representations*.

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.

Akshat Gupta, Dev Sajnani, and Gopala Anumanchipalli. A unified framework for model editing. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15403–15418, 2024.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 8342–8360, 2020.

Tom Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with grace: Lifelong model editing with discrete key-value adaptors. *Advances in Neural Information Processing Systems*, 36:47934–47959, 2023.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Joel Jang, Dongkeun Yoon, Sohee Yang, Sungmin Cha, Moontae Lee, Lajanugen Logeswaran, and Minjoon Seo. Knowledge unlearning for mitigating privacy risks in language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14389–14408, 2023.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

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.

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.

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)*, pp. 13851–13870, 2024.

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.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Haowen Pan, Xiaozhi Wang, Yixin Cao, Zenglin Shi, Xun Yang, Juanzi Li, and Meng Wang. Precise localization of memories: A fine-grained neuron-level knowledge editing technique for llms. In *The Thirteenth International Conference on Learning Representations*, 2025.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. Membench: Towards more comprehensive evaluation on the memory of llm-based agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19336–19352, 2025.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, pp. 63897–63911. PMLR, 2025.

Rubin Wei, Jiaqi Cao, Jiarui Wang, Jushi Kai, Qipeng Guo, Bowen Zhou, and Zhouhan Lin. Mlp memory: A retriever-pretrained memory for large language models. *arXiv preprint arXiv:2508.01832*, 2025.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

CONTENTS

1 Introduction **1**

2 Related Works **2**

3 Methodology **3**

3.1 Preliminaries: Parametric Memory Modules 3

3.2 Memory Attribution: Locating Memory Traces 4

3.3 Memory Writes as Constrained Rank-One Surgery 4

3.4 Memory Lifecycle and Verification 5

3.5 Mixture-of-Memory-Experts (MoME) Expansion 6

4 Experiments **6**

4.1 Experiment Setup 6

4.2 Results and Analysis 7

5 Conclusions and Future Work **9**

A Proof Details **15**

A.1 Proof of Proposition 1 (Insert Invariance and Exactness) 15

A.2 Proof of Proposition 2 (Delete Soundness) 15

A.3 Proof of Proposition 3 (Expansion Scalability) 16

B Additional Implementation Details **16**

B.1 Memory Module Architecture 16

B.2 Probe and Target Construction 16

B.3 Attribution Computation 17

B.4 Key Bank and Null-Space Projection via Truncated SVD 17

B.5 MoME Gating Network 18

B.6 Delete Operation Details 18

B.7 Verification, Journaling, and Rollback 18

C Additional Experimental Results **19**

C.1 End-to-End Evaluation on LongMemEval 19

C.2 Component-wise Construction from Off-the-Shelf Primitives 20

C.3 Causal Validation of Memory Attribution 20

C.4 Memory Footprint Separation and Paraphrase Robustness 21

C.5 Residual Propagation Analysis 22

C.6 Sensitivity to Delete Steps and Scope of the Preservation Guarantee 22

C.7 Deletion Robustness 22

C.8 Long-Horizon Retention 23
C.9 MoME Shard Expansion Timeline 23
C.10 MoME Routing Quality and Router Updates 24
C.11 Sensitivity to MoME Routing (k) 24
C.12 Verification, Rollback, and Backward Auditing 25
C.13 Parametric versus Explicit Memory Storage 25
C.14 Computational Cost Breakdown 26
C.15 Qualitative Examples 26

D Determining Memory Edit Operation **27**

E Limitation Analysis **29**

A Proof Details

A.1 Proof of Proposition 1 (Insert Invariance and Exactness)

Proposition 4 (Insert Invariance and Exactness, restated). *Let $\mathbf{P}_\perp = \mathbf{I} - \mathbf{V}_r\mathbf{V}_r^\top$, where the columns of $\mathbf{V}_r$ form an orthonormal basis of the row space of $\mathbf{K}_0$, and let $\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W}_{\text{ins}}$ with*

$$\Delta\mathbf{W}_{\text{ins}} = \frac{(\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}})(\mathbf{P}_\perp\mathbf{k}_{\text{new}})^\top}{\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2}, \qquad \mathbf{P}_\perp\mathbf{k}_{\text{new}} \neq \mathbf{0}.$$

Then (i) $\mathbf{W}'\mathbf{k}_j = \mathbf{W}\mathbf{k}_j$ for every $\mathbf{k}_j$ in the row space of $\mathbf{K}_0$, and (ii) $\mathbf{W}'\mathbf{k}_{\text{new}} = \mathbf{v}_{\text{new}}$.

Proof. We decompose the proof into three steps.

Step 1: Properties of $\mathbf{P}_\perp$. The matrix $\mathbf{K}_0 \in \mathbb{R}^{n\times d}$ collects the keys of n stored memories. Let its compact SVD be $\mathbf{K}_0 = \mathbf{U}\mathbf{\Sigma}\mathbf{V}_r^\top$, where $\mathbf{V}_r \in \mathbb{R}^{d\times r}$ with $r = \operatorname{rank}(\mathbf{K}_0)$ has orthonormal columns spanning the row space of $\mathbf{K}_0$. With the truncation threshold ϵ_{svd} of Appendix B.4, $\mathbf{V}_r$ spans the numerical row space, and the argument below holds exactly for keys in this span. Since $\mathbf{V}_r^\top\mathbf{V}_r = \mathbf{I}_r$, the projector $\mathbf{P}_\perp$ is symmetric and idempotent:

$$\mathbf{P}_\perp^\top = \mathbf{P}_\perp, \qquad \mathbf{P}_\perp^2 = \mathbf{I} - 2\mathbf{V}_r\mathbf{V}_r^\top + \mathbf{V}_r(\mathbf{V}_r^\top\mathbf{V}_r)\mathbf{V}_r^\top = \mathbf{P}_\perp. \tag{11}$$

For any $\mathbf{k}_j$ in the row space of $\mathbf{K}_0$, we can write $\mathbf{k}_j = \mathbf{V}_r\boldsymbol{\alpha}_j$ for some $\boldsymbol{\alpha}_j \in \mathbb{R}^r$. Therefore:

$$\mathbf{P}_\perp\mathbf{k}_j = (\mathbf{I} - \mathbf{V}_r\mathbf{V}_r^\top)\mathbf{V}_r\boldsymbol{\alpha}_j = \mathbf{V}_r\boldsymbol{\alpha}_j - \mathbf{V}_r(\mathbf{V}_r^\top\mathbf{V}_r)\boldsymbol{\alpha}_j = \mathbf{0}. \tag{12}$$

Step 2: Preservation of stored keys. For any stored key $\mathbf{k}_j$ in the row space of $\mathbf{K}_0$:

$$\Delta\mathbf{W}_{\text{ins}}\mathbf{k}_j = \frac{(\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}})\,\mathbf{k}_{\text{new}}^\top\mathbf{P}_\perp^\top\mathbf{k}_j}{\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2} = \frac{(\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}})\,\mathbf{k}_{\text{new}}^\top(\mathbf{P}_\perp\mathbf{k}_j)}{\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2} = \mathbf{0}, \tag{13}$$

where the last equality uses Step 1. Hence $\mathbf{W}'\mathbf{k}_j = \mathbf{W}\mathbf{k}_j$, which proves (i).

Step 3: Exact insertion. Using symmetry and idempotence, $(\mathbf{P}_\perp\mathbf{k}_{\text{new}})^\top\mathbf{k}_{\text{new}} = \mathbf{k}_{\text{new}}^\top\mathbf{P}_\perp^\top\mathbf{P}_\perp\mathbf{k}_{\text{new}} = \|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2$. Therefore

$$\Delta\mathbf{W}_{\text{ins}}\mathbf{k}_{\text{new}} = (\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}}) \cdot \frac{\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2}{\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|^2} = \mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}}, \tag{14}$$

so $\mathbf{W}'\mathbf{k}_{\text{new}} = \mathbf{v}_{\text{new}}$, which proves (ii). □

Remark (placement of the projection). The projection must act on the key, i.e., on the input side of the update. If a projector is instead applied to the output side of a precomputed rank-one update $\Delta\mathbf{W} = (\mathbf{v}_{\text{new}} - \mathbf{W}\mathbf{k}_{\text{new}})\mathbf{k}_{\text{new}}^\top/\|\mathbf{k}_{\text{new}}\|^2$, the effect on a stored key remains proportional to $\mathbf{k}_{\text{new}}^\top\mathbf{k}_j$, which is nonzero in general, so stored values are not protected. Forming the update from the projected key $\mathbf{P}_\perp\mathbf{k}_{\text{new}}$ yields both properties of the proposition at once.

Remark (degenerate keys). If $\mathbf{P}_\perp\mathbf{k}_{\text{new}} = \mathbf{0}$, the new key lies in the span of the stored keys, and no update of $\mathbf{W}$ can satisfy (i) and (ii) simultaneously. When $\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|$ is small, the update in Eq. (6) has a large norm. MemEdit therefore redirects inserts with $\|\mathbf{P}_\perp\mathbf{k}_{\text{new}}\|/\|\mathbf{k}_{\text{new}}\| < \delta_{\text{key}}$ to a fresh shard (Section 3.5).

A.2 Proof of Proposition 2 (Delete Soundness)

Proposition 5 (Delete Soundness, restated). *Consider the projected gradient-ascent update $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \eta_t\,\mathbf{M}_i\,\mathbf{G}^{(t)}\,\mathbf{P}_\perp^{(i)}$ with $\mathbf{G}^{(t)} = \nabla_{\mathbf{W}}\mathcal{L}(\mathcal{M}_{\theta^{(t)}}(\mathbf{h}_i), p_i^*)$. After T steps, $\mathbf{W}^{(T)}\mathbf{k}_j = \mathbf{W}^{(0)}\mathbf{k}_j$ for every $\mathbf{k}_j$ in the row space of $\mathbf{K}_0^{(-i)}$. For the all-MLP module, the outputs $\mathcal{M}_{\theta^{(T)}}(\mathbf{h}_j)$ of the corresponding memories equal $\mathcal{M}_{\theta^{(0)}}(\mathbf{h}_j)$.*

Proof. We prove the layer-wise statement by induction over gradient-ascent steps.

Base case ($t = 0$): Before any update, the statement holds trivially.

Inductive step: Assume that $\mathbf{W}^{(t)}\mathbf{k}_j = \mathbf{W}^{(0)}\mathbf{k}_j$ for all preserved keys. For any $\mathbf{k}_j$ in the row space of $\mathbf{K}_0^{(-i)}$:

$$\mathbf{W}^{(t+1)}\mathbf{k}_j = \mathbf{W}^{(t)}\mathbf{k}_j + \eta_t \, \mathbf{M}_i \, \mathbf{G}^{(t)}\big(\mathbf{P}_\perp^{(i)}\mathbf{k}_j\big) = \mathbf{W}^{(t)}\mathbf{k}_j, \tag{15}$$

since Step 1 of the proof of Proposition 1, applied to $\mathbf{K}_0^{(-i)}$, gives $\mathbf{P}_\perp^{(i)}\mathbf{k}_j = \mathbf{0}$. Hence $\mathbf{W}^{(t+1)}\mathbf{k}_j = \mathbf{W}^{(0)}\mathbf{k}_j$, completing the induction.

The argument places no restriction on η_t, $\mathbf{M}_i$, $\mathbf{G}^{(t)}$, or the number of steps. It therefore also covers the extended schedule of Appendix B.6 and the paraphrase-verified mode of Appendix C.7. For a single probe, $\mathbf{G}^{(t)} = \boldsymbol{\delta}_t \mathbf{k}_i^\top$, so each step equals $\eta_t \mathbf{M}_i \boldsymbol{\delta}_t (\mathbf{P}_\perp^{(i)}\mathbf{k}_i)^\top$, a rank-one update of the form in Eq. (5).

From the edited layer to the module output. In the all-MLP module, the probe $\mathbf{h}_j$ reaches $\mathbf{W}$ only through its key $\mathbf{k}_j$, which depends only on the layers before l^*. Since those layers and all other parameters are unchanged and $\mathbf{W}\mathbf{k}_j$ is unchanged, every subsequent computation on $\mathbf{h}_j$ is unchanged, and so is $\mathcal{M}_\theta(\mathbf{h}_j)$. The same argument gives the end-to-end form of Proposition 1(i). □

Note on other architectures and on held-out inputs. For Memory Decoder, the input is a token sequence, and the edited projection also processes hidden states at context positions whose keys are not in the key bank. Attention and residual connections in later layers can carry changes at those positions to the probe position. The guarantee therefore holds exactly at the edited layer and approximately at subsequent layers; Appendix C.5 shows that the propagated effect is negligible. In both architectures, the propositions concern memories whose keys are in the key bank. Appendix C.6 measures preservation for held-out inputs.

A.3 Proof of Proposition 3 (Expansion Scalability)

Proposition 6 (Expansion Scalability, restated)**.** *Under the MoME architecture with N shards, each with initial null-space dimension d_{null}, the total editable memory capacity is $\mathcal{O}(N \cdot d_{\text{null}})$.*

Proof. Each successful rank-one insertion adds a key with $\mathbf{P}_\perp \mathbf{k}_{\text{new}} \neq \mathbf{0}$ to the key bank, which increases $\text{rank}(\mathbf{K}_0)$ by one and consumes one dimension of the null space. A single shard with key dimension d and n_0 pretrained keys has initial null-space dimension $d_{\text{null}} = d - \text{rank}(\mathbf{K}_0) \geq d - n_0$. After m insertions into a single shard, the remaining null-space dimension is $d_{\text{null}} - m$. When $d_{\text{null}} - m < \delta \cdot d$, a new shard is spawned with full null-space dimension d_{null}.

With N shards, the total number of insertions before all shards are exhausted is at most $N \cdot (d_{\text{null}} - \lceil \delta \cdot d \rceil)$. Since d_{null} and δ are constants for a given architecture, this scales as $\mathcal{O}(N \cdot d_{\text{null}})$. Because N can grow without bound (each expansion adds a new shard), the total editable capacity is theoretically unlimited. □

B Additional Implementation Details

B.1 Memory Module Architecture

The parametric memory module follows the all-MLP architecture of (Wei et al., 2025). The detailed hyperparameters are listed in Table 6. The edited projection $\mathbf{W} = \mathbf{W}_{l^*}^{(1)}$ maps the 4096-dimensional hidden state (the key) to the 11008-dimensional intermediate layer, so $d = 4096$ and $d_{\text{ff}} = 11008$.

B.2 Probe and Target Construction

Probes and targets are built automatically for all benchmarks, without manual annotation. For MemEdit-Stress, which is derived from CounterFact, the probe is the relation template with the subject filled in (e.g., "The capital of France is"), and the target is the teacher-forced distribution over the object tokens. For MemBench, the probe is the provided question stem and the target is its answer.

For LoCoMo and LongMemEval, the operation controller (Appendix D) produces the probe in the same call that selects the operation, in three steps. (1) *Canonicalization.* The memory is rewritten

Table 6: Memory module architecture hyperparameters.

Hyperparameter	**Value**
Number of MLP layers	4
Hidden dimension	4096
Intermediate dimension	11008
Activation function	GeLU
Input representation layer (backbone depth)	75%
Pretraining loss	$\beta \cdot \mathcal{L}_{\mathrm{KL}} + (1-\beta) \cdot \mathcal{L}_{\mathrm{CE}}$
Loss weight β	0.5
kNN neighbors k	1024
Pretraining corpus	WikiText-103
Pretraining epochs	10
Learning rate (pretraining)	1×10^{-4} (cosine decay)
Batch size (pretraining)	64
Total parameters (per shard)	$\sim$370M

as a self-contained declarative statement. For example, "Ugh, we finally finished the move, Osaka is so much quieter" becomes "The user lives in Osaka." (2) *Probe construction.* The statement is split into a prefix ("The user lives in") and a target span (" Osaka"). The probe $\mathbf{h}_i = f(c_i)$ is the backbone representation at 75% depth of the prefix preceded by a fixed persona and date header, and the same header is used at edit time and read time. (3) *Target distribution.* On the first target token, $p_i^* = (1-\varepsilon)\,\mathrm{onehot}(y_i) + \varepsilon\,\mathbf{1}/|\mathcal{V}|$ over the vocabulary $\mathcal{V}$, with $\varepsilon = 0.01$ matching the label smoothing used in memory pretraining; multi-token spans are handled by teacher forcing. Modify uses the analogous $p_i'^*$ on the new span with the same prefix, so the memory keeps its key. Delete ascends the loss with respect to p_i^* on the old span. On the 200 human-annotated LoCoMo turns described in Appendix D, annotators agreed with the controller's prefix/target split in 94.1% of cases.

B.3 Attribution Computation

The integrated-gradient attribution of Eq. (3) is computed with $S = 20$ Riemann steps, $\alpha_s = s/S$:

$$\mathcal{A}_{l,j}(m_i) \approx \left| \sum_{s=1}^{S} \frac{\partial \mathcal{L}\big(\mathcal{M}_\theta(\tilde{\mathbf{h}}_i(\alpha_s)), p_i^*\big)}{\partial a_{l,j}} \Big(a_{l,j}\big(\tilde{\mathbf{h}}_i(\alpha_s)\big) - a_{l,j}\big(\tilde{\mathbf{h}}_i(\alpha_{s-1})\big) \Big) \right|. \tag{16}$$

The baseline $\bar{\mathbf{h}}$ is the mean hidden representation computed over a random subset of 10K examples from the pretraining corpus, cached once before editing. Computing attribution for a single memory trace takes approximately 12ms on a single A100 GPU, including the forward and backward passes for all 20 interpolation steps.

Attribution scores are computed at every layer of the memory module, and the memory footprint $\mathcal{F}_\tau$ selects neurons globally across all layers. The footprint neurons at layer l^* form the row mask $\mathbf{M}_i$, and the full footprint determines the Query confidence $c_i = \sum_{(l,j)\in\mathcal{F}_\tau(m_i)} \mathcal{A}_{l,j}(m_i) / \sum_{l,j} \mathcal{A}_{l,j}(m_i)$. For numerical stability, we add a small $\epsilon_{\mathrm{attr}} = 10^{-8}$ to the denominator when normalizing attribution scores. Attribution can optionally be averaged over several automatically generated paraphrases of the probe, which makes footprints more robust to rephrased queries at a proportional increase in cost (Appendix C.4).

B.4 Key Bank and Null-Space Projection via Truncated SVD

For each stored memory, the key bank holds its key at layer l^* (a 4096-dimensional fp16 vector, 8KB) and its target distribution restricted to the highest-probability tokens. At 10K memories, the key bank occupies about 80MB, roughly 11% of a 740MB shard. The null-space projector and all preservation terms are computed from this cache, so no original dialogue is retained.

The null-space projection matrix $\mathbf{P}_\perp$ is computed via truncated SVD of the key matrix $\mathbf{K}_0$. We retain singular vectors whose singular values exceed $\epsilon_{\mathrm{svd}} = 10^{-5} \cdot \sigma_{\max}$, where $\sigma_{\max}$ is the largest singular value. The projection is:

$$\mathbf{P}_\perp = \mathbf{I} - \mathbf{V}_r \mathbf{V}_r^\top, \tag{17}$$

where $\mathbf{V}_r$ contains the right singular vectors corresponding to the retained singular values. This form coincides with $\mathbf{I} - \mathbf{K}_0^\top(\mathbf{K}_0\mathbf{K}_0^\top)^{-1}\mathbf{K}_0$ when $\mathbf{K}_0$ has full row rank and remains well defined otherwise. It is computed once per editing batch and cached. For a key dimension of $d = 4096$ with $n = 500$ existing keys, the SVD takes approximately 45ms on CPU. The same routine yields $\mathbf{P}_\perp^{(i)}$ for Delete by excluding $\mathbf{k}_i$, and the ratio $\|\mathbf{P}_\perp \mathbf{k}_{\text{new}}\|/\|\mathbf{k}_{\text{new}}\|$ for the δ_{key} test of Insert.

B.5 MoME Gating Network

The gating network G_ϕ is a single linear layer $\mathbf{W}_g \in \mathbb{R}^{N\times d}$ followed by softmax, where N is the current number of shards. When a new shard is spawned, $\mathbf{W}_g$ is expanded by appending a new row initialized as the mean of the existing rows plus Gaussian noise ($\sigma = 0.01$). The existing rows are left unchanged, and the new logit starts near the average logit, so it initially displaces little routing mass. The gating network is then fine-tuned for 100 steps using Adam with learning rate 5×10^{-4} on a replay buffer of the 1,000 most recent routing examples.

Each routing example consists of a 4096-dimensional fp16 pooled key and its shard label, about 8MB for the whole buffer; no text or token ids are stored. The buffer follows a FIFO policy, and entries associated with committed deletions are purged. Router fine-tuning runs once per expansion, i.e., three times in the 10K-edit experiment. Routing in MoME therefore relies on rehearsal, whereas single-shard MemEdit is rehearsal-free. During routing, we select the top-$k = 1$ shard; for the main results we also experimented with $k = 2$ but found negligible improvement at the cost of doubled memory-module forward passes.

B.6 Delete Operation Details

The gradient-ascent unlearning procedure uses the following schedule:

- Number of steps: $T = 5$
- Initial step size: $\eta_0 = 0.01$
- Decay: cosine annealing, $\eta_t = \eta_0 \cdot \frac{1}{2}(1 + \cos(\pi t/T))$
- Gradient clipping: max norm 1.0
- Null-space projection: $\mathbf{P}_\perp^{(i)}$ applied on the input side of every step, as in Eq. (8)
- Footprint restriction: row mask $\mathbf{M}_i$ applied on the output side of every step

After the T steps, we verify deletion success by checking whether $\text{KL}(p_i^* \| \mathcal{M}_{\theta'}(\mathbf{h}_i)) > \tau_{\text{del}}$, where $\tau_{\text{del}} = 2.0$. If the condition is not met, we continue for up to $T_{\max} = 10$ additional steps with a reduced step size $\eta_0/2$. Proposition 2 covers these additional steps, since it places no restriction on the step size or the number of steps. In the paraphrase-verified mode (Appendix C.7), the condition is checked on every paraphrased probe of m_i instead of the original probe alone. The resulting delta is retained as a tombstone until the deletion is committed (Appendix B.7). At commit time, $\mathbf{k}_i$ is removed from the key bank and the corresponding routing examples are purged from the replay buffer.

B.7 Verification, Journaling, and Rollback

After each write, MemEdit re-probes the edited memory with one forward pass of the memory module (about 3ms). Insert and Modify must return the intended target, and Delete must satisfy the KL condition of Appendix B.6. In addition, the 20 memories whose keys are nearest to the edited key must still match their cached targets within the MP tolerance. If any check fails, the write is rolled back and the operation is re-dispatched.

Each write is journaled as its low-rank factors, i.e., the vectors $\mathbf{Mu}$ and $\mathbf{q}$ of Eq. (5) for every rank-one step. The journal entry also stores a typed record with the operation, the natural-language memory statement, the target entry id, the shard id, the footprint, the timestamp, and the controller's justification. Rollback subtracts the stored delta, $\theta = \theta' - \Delta\theta$. A rank-one delta occupies about 30KB, so a sliding journal of 500 edits requires about 35MB and the full journal of the 10K-edit experiment about 0.7GB. A Delete is first applied as a tombstone: the entry is marked as deleted and

its delta is retained, and the delta is discarded only after a configurable grace window or an explicit confirmation. In the safeguarded configuration, a Delete is additionally triggered only when three controller samples agree (Appendix D). Appendix C.12 evaluates the verification loop and the cost of rollback.

C Additional Experimental Results

C.1 End-to-End Evaluation on LongMemEval

To further validate MemEdit's generalization to diverse long-term memory benchmarks, we evaluate on **LongMemEval** (Wu et al., 2025), which tests five core memory abilities: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention, over 500 questions embedded in scalable user-assistant chat histories (approximately 115K tokens in the standard setting).

Following our treatment of LoCoMo, we use LongMemEval purely as an end-to-end downstream evaluation: the agent processes multi-session dialogues with MemEdit as its parametric memory backend, where operation types and target memories are automatically determined by the LLM backbone (see Appendix D). We do not compute edit-level metrics (ESR, MP, FR) on this benchmark, as it does not provide edit-specific annotations. Instead, we report QA accuracy on each of the five memory ability categories.

Setup. We use the standard LongMemEval setting ($\sim$115K tokens) with the Qwen3-8B backbone. Baselines include the same non-parametric and parametric methods as in the main experiments (Table 1). All methods process the chat history session-by-session in chronological order. We report means and standard deviations over 3 independent runs.

Table 7: QA accuracy (%) on LongMemEval across five memory ability categories (Qwen3-8B backbone). Results are mean $\pm$ std over 3 runs. Best results are in **bold**, and the best baseline is underlined.

Method	Info. Extract.	Multi-Sess.	Temporal	Know. Update	Abstention	Avg.
Non-parametric baselines						
RAG-Base	$38.2_{\pm 1.3}$	$24.5_{\pm 1.5}$	$21.8_{\pm 1.7}$	$28.4_{\pm 1.4}$	$33.1_{\pm 2.0}$	$29.2_{\pm 1.1}$
A-Mem	$42.7_{\pm 1.1}$	$29.3_{\pm 1.2}$	$25.6_{\pm 1.4}$	$31.2_{\pm 1.3}$	$36.8_{\pm 1.8}$	$33.1_{\pm 0.9}$
MemoryOS	$45.3_{\pm 1.0}$	$33.8_{\pm 1.1}$	$28.4_{\pm 1.5}$	$32.7_{\pm 1.2}$	$38.5_{\pm 1.6}$	$35.7_{\pm 0.8}$
Mem0	$40.1_{\pm 1.2}$	$26.7_{\pm 1.4}$	$23.2_{\pm 1.6}$	$29.8_{\pm 1.5}$	$34.6_{\pm 1.9}$	$30.9_{\pm 1.0}$
Parametric baselines						
MLP Memory	$47.1_{\pm 0.9}$	$35.6_{\pm 1.0}$	$30.2_{\pm 1.2}$	$33.5_{\pm 1.1}$	$37.4_{\pm 1.5}$	$36.8_{\pm 0.7}$
Memory Decoder	$46.3_{\pm 1.0}$	$34.2_{\pm 1.1}$	$29.5_{\pm 1.3}$	$32.8_{\pm 1.2}$	$36.9_{\pm 1.6}$	$35.9_{\pm 0.8}$
Editing baselines (adapted)						
AlphaEdit-M	$\underline{48.5}_{\pm 0.8}$	$\underline{36.9}_{\pm 0.9}$	$\underline{31.4}_{\pm 1.1}$	$\underline{38.6}_{\pm 0.9}$	$\underline{39.2}_{\pm 1.3}$	$\underline{38.9}_{\pm 0.7}$
MemEdit (Ours)	$\mathbf{50.3}_{\pm 0.7}$	$\mathbf{39.1}_{\pm 0.8}$	$\mathbf{33.7}_{\pm 1.0}$	$\mathbf{42.8}_{\pm 0.8}$	$\mathbf{41.5}_{\pm 1.2}$	$\mathbf{41.5}_{\pm 0.6}$
MemEdit + MoME	$\mathbf{50.3}_{\pm 0.7}$	$38.8_{\pm 0.9}$	$33.5_{\pm 1.1}$	$42.5_{\pm 0.9}$	$41.3_{\pm 1.3}$	$41.3_{\pm 0.6}$

Analysis. Table 7 reveals several notable findings. First, all methods exhibit substantially lower accuracy on LongMemEval compared to LoCoMo (Table 1), reflecting the significantly longer interaction histories ($\sim$115K vs. $\sim$16K tokens) and more demanding question types in LongMemEval. This is consistent with LongMemEval's original findings that even commercial systems such as GPT-4o achieve only 30%–70% accuracy in similar settings. Second, MemEdit achieves the highest average accuracy (41.5%), outperforming AlphaEdit-M by 2.6% and the best non-parametric baseline (MemoryOS) by 5.8%. The improvement is most pronounced on the *Knowledge Updates* category (+4.2% over AlphaEdit-M), where MemEdit's Modify operation directly addresses the need to revise outdated facts—a capability that static parametric memories and standard retrieval-based methods lack. Third, MemEdit also shows meaningful gains on *Abstention* (+2.3% over AlphaEdit-M), suggesting that the Delete operation helps the agent properly handle retracted or invalidated information rather than hallucinating answers based on erased memories. Non-parametric baselines, which retain all entries unless explicitly purged, tend to produce more false-positive responses on unanswerable questions. Fourth, *Temporal Reasoning* remains the most challenging category for all methods (best: 33.7%), indicating that parametric memory modules still struggle with fine-grained

temporal inference—a direction for future work. Finally, MemEdit + MoME performs comparably to single-shard MemEdit, as the 115K-token history does not exhaust a single shard's null-space capacity. This is expected: the MoME expansion mechanism primarily benefits scenarios with thousands of sequential edits (cf. Table 2), whereas LongMemEval involves on the order of hundreds of memory operations.

C.2 Component-wise Construction from Off-the-Shelf Primitives

MemEdit builds on established primitives, including integrated gradients, null-space projection, gradient-ascent unlearning, and gated mixtures of experts. To measure how much of its performance comes from the way these primitives are adapted and connected, we construct *Naive-Compose*, which assembles the same ingredients off the shelf: IG localization, a null-space projector applied to the output side of a precomputed rank-one update (see the first remark in Appendix A), vanilla gradient ascent for deletion, and a fixed mixture of five experts. We then add MemEdit's design choices one at a time on MemEdit-Stress with the Qwen3-8B backbone.

Table 8: Building MemEdit from off-the-shelf primitives, adding one design choice at a time (MemEdit-Stress, Qwen3-8B backbone). The last row equals MemEdit + MoME in Table 2.

Variant	**ESR@500**	**MP@500**	**ESR@10K**	**MP@10K**
Naive-Compose (off-the-shelf ingredients)	88.9	87.4	69.2	71.5
+ key-side null-space projection	91.6	93.1	76.4	80.2
+ footprint-restricted writes	92.8	95.7	79.1	84.6
+ leave-one-out preserved set (Delete)	93.1	96.0	81.3	87.0
+ capacity-triggered expansion (full MemEdit)	**93.5**	**96.4**	**89.2**	**93.5**

Table 8 shows that every design choice contributes. The two components shared by the write operations, key-side projection and footprint restriction, improve ESR at both horizons, by 2.7 and 7.2 points and by a further 1.2 and 2.7 points, respectively, and together raise MP@500 from 87.4% to 95.7%. Preserving $\mathbf{K}_0^{(-i)}$ during deletion adds further gains, and replacing the fixed mixture with capacity-triggered expansion gives the largest improvement at 10K edits (+7.9 ESR). Overall, Naive-Compose trails full MemEdit by 4.6 ESR at 500 edits and by 20.0 ESR at 10K edits, and the gap widens as edits accumulate.

C.3 Causal Validation of Memory Attribution

Memory Attribution assumes that a small set of neurons is causally responsible for each memory. We test this assumption along two axes, necessity and sufficiency, and compare the IG ranking with an exhaustive causal ranking.

Table 9: Necessity test. Recall after mean-ablating neuron sets of size $\tau = 0.03$ selected by different criteria, and Spearman correlation ρ of each criterion's ranking with an exhaustive leave-one-out causal ranking on a 512-neuron subsample.

Ablated set	**Recall of m_i (%)**	**Recall of 500 others (%)**	**Spearman ρ**
None (control)	96.7	96.4	—
Top-τ, integrated gradients (ours)	**6.3**	92.8	**0.82**
Top-τ, gradient $\times$ activation	22.9	90.6	0.54
Top-τ, activation magnitude	58.4	88.7	0.31
Random-τ	91.5	94.1	0.01
Bottom-τ	96.1	96.2	—

Necessity. For each target memory m_i, we mean-ablate the top-τ attributed neurons ($\tau = 0.03$) and measure recall of m_i and of 500 other memories. As shown in Table 9, ablating the IG footprint reduces recall of m_i from 96.7% to 6.3%, while recall of the other memories remains at 92.8%. Footprints selected by gradient $\times$ activation or by activation magnitude are less effective at removing

the memory and cause more collateral damage. Random and bottom-ranked neurons have almost no effect.

Agreement with the causal ranking. On a 512-neuron subsample, exhaustive leave-one-out ablation is tractable and gives a ground-truth causal ranking. The last column of Table 9 shows that the IG ranking correlates with it at $\rho = 0.82$, compared with 0.54 for gradient $\times$ activation, 0.31 for activation magnitude, and 0.01 for a random ranking.

Table 10: Sufficiency test. Recall of m_i after corrupting layer l^* and restoring only the selected neurons to their original weights.

Restored set	None	Random-τ	Bottom-τ	Top-τ (ours)	Top-2τ	All (ceiling)
Recall of m_i (%)	3.1	8.7	4.4	83.5	91.2	96.7

Sufficiency. We corrupt the weights of layer l^* and restore only a selected subset of its neurons to their original values. Table 10 shows that restoring the top-τ footprint recovers 83.5% recall of m_i, i.e., 86% of the 96.7% ceiling, while restoring random or bottom-ranked neurons recovers almost nothing.

Concentration. Memory traces are concentrated but not strictly sparse. The top 0.5%, 1%, 3%, 5%, and 10% of neurons carry 21.4%, 34.8%, 61.7%, 72.3%, and 84.1% of the total attribution mass, respectively. The mass outside the footprint explains why deletion does not reach 100% forgetting and why part of a deleted memory remains recoverable under paraphrased probes (Appendix C.7).

C.4 Memory Footprint Separation and Paraphrase Robustness

We analyze how well footprints separate distinct memories and how stable they remain under paraphrase. For 200 randomly sampled memory pairs (m_i, m_j) with $i \neq j$, we compute the Jaccard similarity $J(\mathcal{F}_\tau(m_i), \mathcal{F}_\tau(m_j))$ between their footprints. For 200 memories, we also generate 10 paraphrased probes each, verified by NLI entailment, and compute the mean Jaccard similarity between the footprints of paraphrases of the same memory.

Table 11: Jaccard similarity between memory footprints. Distinct memories: mean $\pm$ std over 200 pairs (lower is better). Same memory: mean over 10 paraphrased probes for each of 200 memories (higher is better). The separation ratio divides the second row by the first.

Sparsity τ	0.01	0.03	0.05	0.10
Distinct memories $\downarrow$	$0.002_{\pm 0.003}$	$0.008_{\pm 0.006}$	$0.021_{\pm 0.012}$	$0.073_{\pm 0.031}$
Same memory, 10 paraphrases $\uparrow$	0.71	0.83	0.87	0.92
Separation ratio $\uparrow$	355$\times$	104$\times$	41$\times$	13$\times$

As shown in Table 11, at the default $\tau = 0.03$, distinct memories overlap by only 0.008 on average, whereas paraphrases of the same memory overlap by 0.83, a 104$\times$ separation. This supports the assumption underlying our CRUD operations that individual memories are localized to largely non-overlapping neuron subsets. Paraphrase stability follows from the input representation: at 75% backbone depth, paraphrases of the same statement have a mean cosine similarity of 0.89, compared with 0.31 for distinct memories. At larger $\tau = 0.10$, overlap between distinct memories increases substantially, which explains the degraded MP observed in the ablation study (Table 4 in the main text).

Stable footprints do not fully guarantee that an edit generalizes to rephrased queries. Table 12 evaluates edits on 10 held-out paraphrases per memory. With single-probe attribution, ESR decreases from 94.6% on the original probe to 87.3% on paraphrases. Averaging attribution over automatically generated paraphrases recovers most of this gap: with three paraphrases, paraphrase ESR reaches 92.6% while MP stays at 97.1%, and with five paraphrases it reaches 93.9%. Averaging over three paraphrases raises the attribution time from 12.3ms to about 34ms, and attribution accounts for roughly a quarter of a full operation.

Table 12: Edit quality under paraphrase for different attribution probe sets. ESR is measured on the original probe and on 10 held-out paraphrases per memory.

Attribution probe set	**ESR, original (%)**	**ESR, paraphrases (%)**	**MP (%)**
Single probe (default)	94.6	87.3	97.3
Union of 3 paraphrase footprints	94.4	91.8	96.9
Mean attribution over 3 paraphrases	94.8	92.6	97.1
Mean attribution over 5 paraphrases	95.0	93.9	96.8

C.5 Residual Propagation Analysis

For the all-MLP module, Propositions 1 and 2 imply end-to-end invariance, because a probe reaches the edited projection only through its own key. For Memory Decoder, the edited projection also processes hidden states at other context positions, whose keys are not in the key bank, and attention and residual connections in later layers can carry these changes to the probe position. To quantify this effect, we measure the KL divergence of preserved memory outputs at each layer after editing layer 2 of Memory Decoder. Table 13 shows that the perturbation at the edited layer is exactly zero, by construction, and the propagated effect remains negligible (< 0.005 KL divergence) across all subsequent layers.

Table 13: KL divergence of preserved memory outputs at each layer after a single edit (Memory Decoder architecture, averaged over 100 edits).

Layer	**1**	**2 (edited)**	**3**	**4**	**Output**
KL divergence	0.000	0.000	$0.002_{\pm 0.001}$	$0.003_{\pm 0.001}$	$0.004_{\pm 0.002}$

C.6 Sensitivity to Delete Steps and Scope of the Preservation Guarantee

We vary the number of gradient-ascent steps T for the Delete operation and measure both forgetting ratio (FR) and memory preservation (MP) on 100 delete operations. The results in Table 14 reveal a clear trade-off: more steps improve forgetting completeness but gradually erode preservation of unrelated memories.

Table 14: Effect of delete steps T on Forgetting Ratio and Memory Preservation (100 delete operations). MP is also split by whether a preserved memory's key is in the key bank $\mathbf{K}_0$ (roughly half of the evaluation set); the split was not computed for $T = 15$.

Steps T	1	3	5	10	15	20
FR (%) ↑	$62.3_{\pm 2.8}$	$82.7_{\pm 1.9}$	$91.4_{\pm 1.0}$	$95.8_{\pm 0.7}$	$96.2_{\pm 0.6}$	$96.4_{\pm 0.5}$
MP, overall (%) ↑	$98.9_{\pm 0.2}$	$97.8_{\pm 0.3}$	$96.1_{\pm 0.6}$	$94.2_{\pm 0.8}$	$92.8_{\pm 1.1}$	$91.5_{\pm 1.3}$
MP, keys in $\mathbf{K}_0$ (%) ↑	100.0	100.0	99.98	99.95	—	99.91
MP, held-out keys (%) ↑	97.8	95.6	92.2	88.4	—	83.1

To locate the source of this erosion, we split the preserved evaluation set by whether a memory's key is in the key bank. For keys in $\mathbf{K}_0$, MP stays at or above 99.9% for all T, consistent with Proposition 2; the small deviation from 100% reflects floating-point accumulation over steps. The erosion comes almost entirely from held-out keys, which the projection does not constrain and whose MP falls from 97.8% at $T = 1$ to 83.1% at $T = 20$. Enlarging the key bank therefore directly improves preservation of held-out inputs, at a cost of 8KB per additional key. We select $T = 5$ as the default, which achieves >91% FR with >96% MP.

C.7 Deletion Robustness

Low confidence on the original probe does not by itself show that a memory has been removed. We therefore evaluate 300 deleted memories on MemBench (Qwen3-8B backbone) under several

probe families and compare against a shard retrained without the memory, which serves as the gold standard. The probe families are the original probe, 10 paraphrases per memory, indirect or entailed queries, and adversarial extraction through prefix injection, role-play framing, and many-shot pattern completion. We also measure relearning, i.e., the number of gradient steps on a handful of examples needed to restore 50% recall, which distinguishes removal from output suppression, and a loss-based membership inference attack (MIA), for which an AUC of 0.5 is ideal.

Table 15: Deletion robustness on 300 deleted memories (MemBench, Qwen3-8B backbone). FR is reported per probe family; the retrained shard is the gold standard.

Probe family	**MemEdit** ($T = 5$)	**MemEdit** ($T = 10$)	**Grad-ascent only**	**AlphaEdit-M**	**Retrained (gold)**
Original probe FR (%) ↑	91.4	95.8	88.1	88.3	100.0
Paraphrase (10 per memory) FR (%) ↑	84.7	92.3	71.5	76.9	100.0
Indirect / entailed FR (%) ↑	79.2	88.6	63.8	70.4	99.6
Adversarial extraction FR (%) ↑	76.5	86.9	58.2	66.1	99.4
Relearn steps to 50% recall ↑	34	61	9	17	78
Loss-based MIA AUC (0.5 ideal)	0.61	0.56	0.74	0.69	0.50
MP after delete (%) ↑	96.1	94.2	81.3	90.7	n/a

Table 15 reports the results. FR declines as probes move further from the original phrasing; at the default $T = 5$, it is 84.7% under paraphrase and 76.5% under adversarial extraction. MemEdit nevertheless outperforms gradient ascent alone and AlphaEdit-M under every probe family, and among the editing methods its relearning resistance and MIA AUC are the closest to the retrained model, indicating that traces are reduced rather than merely masked. The number of steps is a practical operating knob: $T = 10$ raises adversarial FR to 86.9% at 94.2% MP. An adaptive, paraphrase-verified mode that continues ascent until all paraphrased probes exceed τ_{del} (mean $T = 8.3$) reaches 97.2%, 96.4%, and 89.1% FR on original, paraphrased, and adversarial probes, respectively, with 95.0% MP and 79.4ms latency. These results show that Delete provides practical, measurable erasure from parametric memory, but not certified unlearning (Appendix E).

C.8 Long-Horizon Retention

Per-edit MP does not show whether the earliest memories survive a long edit stream. We therefore track recall of the first 100 memories inserted into the MemEdit-Stress stream as edits accumulate. As shown in Table 16, MemEdit + MoME retains 94.3% of these memories after 10K edits, compared with 82.7% for single-shard MemEdit and 71.9% for AlphaEdit-M. Because the key bank keeps the keys of all stored memories, the oldest entries remain protected by the projection throughout the stream, and expansion prevents the active shard from saturating. Appendix C.12 complements this analysis with a backward audit of all prior entries under the automatic controller.

Table 16: Recall (%) of the first 100 inserted memories as sequential edits accumulate (MemEdit-Stress, Qwen3-8B backbone).

Method	**1K edits**	**3K edits**	**5K edits**	**10K edits**
AlphaEdit-M	92.5	86.7	80.1	71.9
MemEdit (single shard)	96.8	93.1	89.4	82.7
MemEdit + MoME	**98.1**	**97.2**	**96.0**	**94.3**

C.9 MoME Shard Expansion Timeline

Table 17 shows the null-space rank and shard count over the course of 10K sequential edits. Each time the null-space rank of the active shard drops below the threshold ($\delta = 0.15$), a new shard is spawned and subsequent edits are routed to the shard with the most available capacity. The final null-space rank of 45.1% indicates that the system still has substantial capacity for future edits without requiring further expansion, confirming the scalability of the MoME strategy. In addition, the δ_{key} test redirects inserts whose keys are nearly spanned by stored keys. It fires for 0.9% of inserts at 1K edits and for 7.3% at 10K edits, rising as shards fill up, and thus signals the same capacity pressure that drives expansion.

Table 17: MoME expansion timeline during the 10K-edit scalability experiment (Qwen3-8B backbone).

Event	Edit #	Null-space rank before	Total shards after
Initial	0	3,482 / 4,096	2
Expansion 1	$\sim$1,200	608 / 4,096 (14.8%)	3
Expansion 2	$\sim$3,800	612 / 4,096 (14.9%)	4
Expansion 3	$\sim$7,500	598 / 4,096 (14.6%)	5
Final	10,000	1,847 / 4,096 (45.1%)	5

C.10 MoME Routing Quality and Router Updates

We measure routing quality by Recall@k, the fraction of queries whose target memory resides in one of the top-k shards selected by G_ϕ. Table 18 (top) shows that during the 10K-edit experiment, Recall@1 decreases only from 99.4% with two shards to 97.1% with five, and Recall@2 remains above 99%. To study larger mixtures, we run a routing scaling study with up to 128 shards (bottom). The single linear gate degrades beyond about 32 shards and reaches 75.6% Recall@1 at 128 shards, whereas a two-layer MLP gate and a hierarchical two-level router maintain 89.8% and 94.6%, respectively. Writes that are misrouted during editing are additionally caught by the verification probe (Appendix B.7).

Table 18: Routing quality of the MoME gate. Top: Recall@k (%) during the 10K-edit experiment. Bottom: Recall@1 (%) of different gate architectures in a scaling study with up to 128 shards.

Number of shards N	2	3	4	5
Recall@1	99.4	98.7	97.9	97.1
Recall@2	100.0	99.8	99.5	99.2

Number of shards N	5	16	32	64	128
Linear gate (default)	97.1	93.8	89.4	83.1	75.6
2-layer MLP gate	97.4	96.2	94.7	92.3	89.8
Hierarchical 2-level router	97.3	96.8	96.1	95.4	94.6

Table 19 examines catastrophic forgetting in the gate itself. Fine-tuning the gate without replay after each expansion lowers Recall@1 on the old shards to 88.3% and ESR at 10K edits to 84.1%. The replay buffer of Appendix B.5, which is the default, restores old-shard Recall@1 to 97.4%. Anchoring the pre-existing rows of $\mathbf{W}_g$ with an L2-SP penalty and sampling the buffer in a class-balanced way give further small gains, up to 90.1% ESR at 10K edits.

Table 19: Effect of the gate update strategy on routing Recall@1 (%) and ESR at 10K edits (%).

Gate update strategy	**Recall@1, old shards**	**Recall@1, new shard**	**ESR@10K**
Fine-tuning without replay	88.3	96.2	84.1
+ replay buffer (default)	97.4	96.8	89.2
+ L2-SP anchor on existing rows	98.2	96.5	89.7
+ class-balanced sampling	98.6	96.9	90.1

C.11 Sensitivity to MoME Routing (k)

In Table 20, we compare top-k routing with $k \in \{1, 2, 3\}$ for the MoME gating network on 500 mixed edits (MemBench, Qwen3-8B backbone). Increasing k yields marginal ESR gains (+0.4% from $k = 1$ to $k = 3$) at the cost of linearly increasing latency, since each additional shard requires a full forward pass through the memory module. We use $k = 1$ as the default for its best efficiency-accuracy trade-off.

Table 20: Effect of MoME top-k routing on ESR, MP, and inference latency.

Top-k	**ESR (%)**	**MP (%)**	**Latency (ms/query)**
$k = 1$	94.6±0.5	97.3±0.3	8.2±0.4
$k = 2$	94.9±0.4	97.1±0.4	14.7±0.6
$k = 3$	95.0±0.5	96.8±0.5	21.3±0.9

C.12 Verification, Rollback, and Backward Auditing

We evaluate the verification loop of Section 3.4 by injecting random wrong operation types into the controller output on LoCoMo. As shown in Table 21, without safeguards, LoCoMo F1 falls from 46.2 to 39.6 at a 20% injected error rate. With verification and rollback, it falls only to 43.8, and 85.9% of the erroneous edits are caught. At a 0% injected error rate, the two configurations give identical F1.

Table 21: Robustness to injected controller errors on LoCoMo (Qwen3-8B backbone).

Injected controller error rate	0%	5%	10%	20%
LoCoMo F1, no safeguard	46.2	44.8	43.1	39.6
LoCoMo F1, verification + rollback	46.2	45.9	45.3	43.8
Erroneous edits caught (%)	n/a	88.4	87.1	85.9

Table 22 lists the storage and rollback cost of each journaled write. Rollback takes at most 2.1ms and restores the pre-edit state up to floating-point error.

Table 22: Journal storage and rollback cost per write.

Operation	**Storage**	**Rollback latency**	**Post-rollback KL to pre-edit state**
Insert (rank 1)	30KB	0.9ms	0.000
Modify (rank 1 on footprint rows)	30KB	1.4ms	3×10^{-7}
Delete (rank $\leq T = 5$)	151KB	2.1ms	2×10^{-6}

Forward evaluation cannot reveal a wrong-target Modify that silently damages an earlier memory. We therefore run a backward audit every 100 edits, re-probing all prior entries from their cached keys and targets without access to the original dialogue. Each regression is classified as an intended supersession, collateral interference, or a silent invalidation, i.e., damage to a non-target entry caused by a wrong-target Modify. The silent invalidation rate (SIR) is the fraction of previously written entries that are silently invalidated at a checkpoint. Since SIR is zero by construction when operations are taken from ground-truth labels, we compute it on an edit stream whose operations are selected by the automatic controller. SIR reaches 0.9%, 1.7%, and 2.2% at 1K, 5K, and 10K edits, and verification with rollback reduces it to 0.1%, 0.2%, and 0.3%. Mechanism-level interference is measured separately by MP. Re-probing costs 8.2ms per entry, about 82s per sweep over 10K entries, and runs asynchronously.

C.13 Parametric versus Explicit Memory Storage

Explicit stores such as plaintext files, databases, knowledge graphs, and vector stores are human-readable and support hard deletion, which makes them the natural choice for enumerable, low-volume, human-audited memories such as project notes, user preferences, and task histories. Parametric memory becomes attractive when the memory is large or diffuse enough for top-K retrieval to truncate relevant context, when retrieval sits on the critical path of latency-constrained serving, when the knowledge is statistical rather than enumerable (e.g., style, domain priors, or procedural habits), and when on-device deployment lacks index infrastructure. The long-history benchmarks reflect this regime: on LongMemEval, MemEdit reaches 41.5% average accuracy against 35.7% for the best non-parametric baseline (Appendix C.1).

To locate the crossover, we compare a top-8 vector store with MemEdit on synthetic memory corpora of increasing size (Qwen3-8B backbone). Absolute values differ from Table 1 because entries are sampled uniformly.

Table 23: Explicit versus parametric memory as the number of stored entries grows (synthetic corpora, Qwen3-8B backbone).

Memory entries	**1K**	**10K**	**100K**
Vector store (top-8), QA accuracy (%)	78.4	73.1	64.9
Vector store (top-8), per-turn latency	14ms	26ms	68ms
MemEdit, QA accuracy (%)	77.9	76.8	75.2
MemEdit, per-turn latency	8.2ms	8.2ms	8.5ms

Table 23 shows that the vector store is slightly more accurate at 1K entries, but its accuracy declines under retrieval truncation and its latency grows from 14ms to 68ms per turn. MemEdit stays nearly flat in latency and degrades more slowly, so the crossover lies between 1K and 10K entries. The two paradigms are complementary: the edit journal of Section 3.4 makes changes to parametric memory auditable and reversible, and MemEdit can serve as the parametric component of hybrid systems such as MemOS (Li et al., 2025).

C.14 Computational Cost Breakdown

Table 24 details the wall-clock time and GPU memory for each component of a single operation. The dominant cost is the LLM backbone forward pass to extract $\mathbf{h}$, which is shared across all operation types and is needed for generation anyway. The editing-specific computation (attribution and parameter update) is lightweight, because the edited object is the 370M-parameter memory module rather than the backbone. All timings use CUDA events on an exclusive GPU, with 100 warm-up iterations followed by 1,000 timed iterations.

Table 24: Computational cost per operation (single A100 80GB, Qwen3-8B backbone, batch size 1).

Component	**Time (ms)**	**GPU Memory (MB)**
LLM forward (extract $\mathbf{h}$)	$18.4_{\pm 1.2}$	15,200
Attribution ($S = 20$ steps)	$12.3_{\pm 0.8}$	1,480
SVD for $\mathbf{P}_\perp$ (cached)	$2.1_{\pm 0.3}$	134
Insert (rank-one update)	$1.8_{\pm 0.2}$	67
Modify (least-squares solve)	$8.5_{\pm 0.6}$	312
Delete ($T = 5$ grad-ascent steps)	$19.7_{\pm 1.4}$	1,520
MoME gating	$0.3_{\pm 0.1}$	4
Total (Insert)	**34.9**	—
Total (Modify)	**41.6**	—
Total (Delete)	**52.8**	—

The totals include attribution for every write: Modify and Delete need the footprint before editing, and Insert records the footprint of the new memory afterwards. Turns classified as None, which make up more than half of all dialogue turns (Appendix D), incur only the memory read (8.2ms, Table 20) and the controller call. Amortized over the LoCoMo operation distribution, MemEdit adds about 21ms per turn on top of the backbone forward pass, compared with 14–68ms for vector-store retrieval alone (Appendix C.13).

C.15 Qualitative Examples

Table 25 shows representative examples of each CRUD operation and the resulting memory behavior.

Table 25: Qualitative examples of MemEdit CRUD operations.

Op.	Input	Before Edit	After Edit
Insert	"User started learning piano in March 2025"	No relevant output (uniform distribution)	Correctly recalls piano-related context with high confidence (0.87)
Modify	"User moved from Tokyo to Osaka"	"User lives in Tokyo" (confidence 0.91)	"User lives in Osaka" (confidence 0.89); unrelated memory "User likes sushi" unchanged (0.93)
Delete	"Remove user's medical history"	"User was diagnosed with..." (confidence 0.85)	Output degrades to near-uniform (confidence 0.04); related memory "User visits hospital monthly" partially preserved (0.72)
Query	"What does the module know about user's pets?"	—	Returns top-5 attributed neurons in layers 2–3 with attribution scores [0.31, 0.24, 0.18, 0.12, 0.08]

D Determining Memory Edit Operation

Prompt. For LoCoMo and LongMemEval, where edit operations are not pre-specified by the benchmark, we prompt the LLM backbone to automatically determine the operation type, the target memory, and the probe for each incoming dialogue turn. The prompt template is shown below.

Memory Operation Selection Prompt

```
You are the memory controller of a personal AI assistant. Given a
new dialogue turn from the ongoing conversation, decide whether and
how to update the assistant's parametric memory.

== Stored Memory Entries ==
{memory_entries}

== New Dialogue Turn ==
Session: {session_id}, Turn: {turn_id}
Timestamp: {timestamp}
Speaker: {speaker}
Content: "{utterance}"

== Task ==
Analyze the new turn and select exactly ONE of the following
operations:

INSERT: The turn introduces factual information (personal details,
preferences, events, plans, or stated facts) that is genuinely
new--not present in or inferable from any existing memory entry.
Specify the factual content to memorize.

MODIFY: The turn updates, corrects, or supersedes an existing
memory entry. For example, the user previously said "I live in
Tokyo" and now says "I actually moved to Osaka last month." Specify
the previous content to modify and the corrected content.

DELETE: The turn explicitly retracts, denies, or requests removal
of a previously stated fact. For example, "Please forget what
I told you about my medical history" or "That's no longer true,
ignore it." Specify the previous content to delete.

NONE: The turn is a greeting, acknowledgment, follow-up question,
emotional expression, or casual conversation that does not contain
new factual content worth memorizing.

== Guidelines ==
```

```
- Only INSERT genuinely new facts; do not memorize opinions, filler,
or pleasantries.
- Prefer MODIFY over INSERT when the new information overlaps with
or refines an existing entry, to avoid duplicates.
- Use DELETE only when there is an explicit retraction or removal
request, not merely a topic change.
- When uncertain between INSERT and NONE, prefer INSERT to avoid
information loss.
- When uncertain between MODIFY and INSERT, prefer MODIFY to
maintain memory consistency.
- Write every memory as a self-contained declarative sentence (e.g.,
"The user lives in Osaka."), and split it into a probe prefix and a
target span (e.g., "The user lives in" / " Osaka").

Respond with a JSON object and nothing else:

{
  "operation": "<INSERT | MODIFY | DELETE | NONE>",
  "target memory": "<existing memory contents
    to modify or delete; null for INSERT and NONE>",
  "new/updated memory": "<factual content to insert, or the updated
    content after modification; null for DELETE and NONE>",
  "probe": {"prefix": "<statement prefix>",
            "target": "<target span>"},
  "reason": "<one-sentence justification>"
}
```

Implementation details. At each dialogue turn, the LLM backbone receives the above prompt and returns a JSON response specifying the selected operation, which is then dispatched to Algorithm 1. The field `{memory_entries}` lists the natural-language statements of the stored memories, read from the edit journal (Appendix B.7). The probe fields give the prefix and target span used to build $\mathbf{h}_i$ and p_i^* (Appendix B.2); they refer to the new or updated memory for Insert and Modify and to the target memory for Delete. Turns classified as None are skipped without invoking any memory operation. For Modify and Delete, the Query confidence c_i of the named entry is checked first. If it is below 0.2, the entry is treated as absent: a Modify is converted into an Insert, and a Delete is skipped. This conversion applies to 6.2% of Modify decisions on LoCoMo. In the safeguarded configuration evaluated in Appendix C.12, a Delete additionally requires agreement among three controller samples; otherwise, the turn falls back to None and is logged.

Further Analysis. We analyze the operation distribution across both benchmarks in Table 26. The majority of dialogue turns contain casual conversation or acknowledgments and are classified as None. Among actionable turns, Insert dominates as conversations progressively introduce new information, while Modify accounts for 8–11% of turns and Delete is rare (<3%), reflecting the natural distribution of conversational dynamics where explicit retractions are uncommon.

Table 26: Distribution of automatically selected operations across dialogue turns.

Benchmark	Insert	Modify	Delete	None
LoCoMo	33.7%	8.5%	1.6%	56.2%
LongMemEval	31.2%	11.3%	2.1%	55.4%

LongMemEval shows a slightly higher Modify ratio (11.3% vs. 8.5%) because its longer interaction histories contain more instances of updated user information across sessions, and a slightly higher Delete ratio (2.1% vs. 1.6%) due to its inclusion of explicit knowledge retraction scenarios.

The classification step adds negligible overhead (<5ms per turn) as it reuses the hidden representations already computed by the backbone's forward pass. To validate the quality of automatic operation selection, we randomly sampled 200 dialogue turns from LoCoMo and had two annotators independently label the appropriate operation type and target entry. Averaged over three controller runs, the LLM backbone achieves 91.3% agreement with human annotations on operation type (Co-

hen's $\kappa = 0.86$) and 87.6% agreement on target entry identification, confirming that the automated classification is sufficiently reliable for end-to-end evaluation.

Error analysis. The controller has no formal correctness guarantee, so we characterize its errors and their consequences. Table 27 gives the confusion matrix of one controller run on the 200 annotated turns, in which 183 turns are classified correctly. Errors are dominated by confusion between Insert and Modify. A Modify misread as Insert (3.0% of turns) leaves a stale duplicate entry, which later appears among the stored entries but is not merged automatically. An Insert misread as Modify (2.0%) or a None misread as Modify (0.5%) overwrites the wrong entry; these cases are handled by the confidence check, verification, and rollback. A None misread as Insert (2.5%) adds a redundant entry that only consumes null-space capacity, and an Insert misread as None (0.5%) loses information without automatic recovery. Delete is never fired spuriously in this sample. On a separate set of 300 turns containing retractions, Delete reaches 97.4% precision and 93.8% recall.

Table 27: Confusion matrix of operation selection on 200 human-annotated LoCoMo turns (rows: human label; columns: controller prediction; one controller run, 183/200 correct).

Human \ Predicted	**Insert**	**Modify**	**Delete**	**None**
Insert (67)	62	4	0	1
Modify (17)	6	11	0	0
Delete (3)	0	0	3	0
None (113)	5	1	0	107

Controller refinements. The controller can be strengthened further. Table 28 adds three refinements in turn: marking the five stored entries whose keys are most similar to the current turn as likely targets (one matrix product over the key bank, 0.4ms), self-consistency over three samples with abstention to None on disagreement, and a two-stage prompt that first decides whether the turn is actionable and then selects the operation type. Together, they raise operation accuracy from 91.3% to 95.6% and target-entry accuracy from 87.6% to 93.1%, bringing LoCoMo F1 to 47.6, within 0.8 points of the 48.4 obtained with oracle operation labels. The main experiments use the single-prompt controller.

Table 28: Effect of controller refinements on operation selection and downstream LoCoMo F1 (Qwen3-8B backbone). Each row adds one refinement to the row above.

Controller configuration	**Op-type acc. (%)**	**Target-entry acc. (%)**	**LoCoMo F1**
Single prompt (default)	91.3	87.6	46.2
+ top-5 candidate entries	93.8	91.2	46.9
+ self-consistency (3 samples, abstain to None)	94.9	92.4	47.3
+ two-stage selection	95.6	93.1	47.6
Oracle operation labels (upper bound)	100.0	100.0	48.4

E Limitation Analysis

While MemEdit demonstrates strong performance across multiple benchmarks, we identify several limitations that merit discussion and suggest directions for future improvement.

Single-layer editing and key-bank coverage. The write operations update a single projection at layer l^* of the memory module. For the all-MLP module, the preservation guarantees then hold end to end, but for Memory Decoder, changes at context positions outside the key bank can reach the probe through later layers, resulting in small but non-zero KL divergence ($<$0.005) at the output (Appendix C.5). The guarantees also cover only memories whose keys are in the key bank, and preservation of held-out inputs declines under long deletion schedules (Appendix C.6). These effects may accumulate under a large number of sequential edits, partially explaining the gradual ESR decline observed even with MoME (Table 2). Extending the null-space projection to jointly cover multiple layers and enlarging the key bank are promising directions, at additional computational and memory cost.

Dependence on operation selection quality. For benchmarks without pre-specified edit labels (LoCoMo, LongMemEval), MemEdit relies on the LLM backbone to classify each dialogue turn and identify the corresponding memory contents (Appendix D). The controller has no formal correctness guarantee, and its errors, mostly confusion between Insert and Modify, can create duplicate entries or overwrite the wrong entry. Verification and rollback catch most erroneous writes and keep silent invalidation at 0.3% of entries after 10K edits (Appendix C.12), but a missed Insert cannot be recovered automatically. The quality of this upstream controller is bounded by the backbone's reasoning ability and may degrade on ambiguous or implicit information updates that lack explicit linguistic cues.

Scalability of null-space computation. The null-space projection matrix $\mathbf{P}_\perp$ requires computing the SVD of the key matrix $\mathbf{K}_0 \in \mathbb{R}^{n \times d}$, which has $O(nd^2)$ complexity. As the number of stored memories n grows within a single shard, this computation becomes increasingly expensive. While MoME mitigates this by distributing memories across shards, extremely large-scale deployments (e.g., $n > 10^5$ per shard) may require approximate null-space methods such as randomized SVD or incremental SVD updates, which we have not explored.

Limited modality coverage. MemEdit currently operates exclusively on textual memories encoded as hidden representations from the LLM backbone. Extending the framework to multimodal parametric memories, where visual, auditory, or structured knowledge is jointly encoded, would require rethinking the attribution mechanism and the definition of memory footprints across heterogeneous representation spaces.

Formal privacy guarantees for Delete. The Delete operation erases targeted memory traces empirically, with 91.4% forgetting on the original probe and 76.5% under adversarial extraction (Appendix C.7), but it does not provide formal privacy guarantees comparable to differential privacy or certified machine unlearning. Relearning and membership inference show that residual traces remain in the parameter space, particularly for memories that overlap with preserved entries. MemEdit should therefore not be treated as a standalone compliance mechanism, and establishing rigorous privacy bounds for the Delete operation, a requirement for compliance with regulations such as GDPR, remains an important open problem.

MoME routing at large shard counts. Our editing experiments use at most $N = 5$ shards. In a routing-only scaling study, the single linear gate degrades beyond about 32 shards, whereas a hierarchical two-level router retains 94.6% Recall@1 with 128 shards (Appendix C.10). End-to-end editing at such scales, where routing errors could apply edits to incorrect shards and degrade both ESR and MP, remains to be evaluated. In addition, MoME routing rehearses on a small buffer of pooled keys, whereas single-shard MemEdit is rehearsal-free.