MEMORILLA: LATENT SEMANTIC MEMORY FOR LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory enables large language models (LLMs) to use information beyond their parameters, such as past user conversations, long documents, or an agent's earlier actions. For a frozen LLM, access to this accumulated knowledge is limited to what a memory system places in its context window. However, gathering and combining relevant information needed for each query while keeping the model's context short remains challenging for existing systems. Here we introduce MEMORILLA, a memory system that gathers information from an entire document collection into a fixed number of query-dependent 'memory tokens' for a frozen LLM. MEMORILLA encodes documents (such as user chats, book chapters) once into reusable embeddings, then learns to retrieve and place query-specific information in the LLM's input embedding space. Using only 16 memory tokens, MEMORILLA achieves the best performance against baselines including Mem0 and MemoRAG on five of eight benchmarks spanning personalization, long-document and knowledge-intensive question answering. Notably, on FactKG, it reaches 84.33% accuracy versus 80.43% with full-context inference, while using 108× fewer tokens to represent the source documents. Furthermore, MEMORILLA can facilitate continual learning in agentic systems where the document count (the agent's past turns) increases throughout the trajectory. In a TextWorld environment, we show that MEMORILLA reduces the agent's tendency to wander between rooms it has already explored, and improves its average success rate from 18.6% with full-history in-context learning to 24.2%. Overall, MEMORILLA aids frozen LLMs in accumulating and using knowledge effectively across long-context interactions and takes a step towards building continually learning AI systems.

1 INTRODUCTION

Memory is central to how large language models (LLMs) are used. Personal assistants must recall a user's history across months of conversations, question-answering systems must reason over documents far longer than the model's context window, and agents must act on everything they have observed. None of this information exists in the model's parameters, and a frozen LLM can use it only through its context (Hu et al., 2025). A *memory system* bridges this gap by encoding accumulated information, storing it persistently, and retrieving what each query needs into the context (Wu et al., 2025). The central challenge for such a system is to give a frozen LLM access to an entire, growing collection of knowledge while keeping its context short (An et al., 2025; Du et al., 2025).

Solving this challenge determines whether LLMs can accumulate knowledge effectively after deployment. In the real world, user preferences change, policies are revised and new findings overturn existing knowledge. However, retraining a model for every change is impractical, and whatever a frozen LLM learns within a session is lost once the text leaves its context (Yao et al., 2023; Shinn et al., 2023). This loss is costly for agents, whose sessions spent debugging a codebase or failed attempts at a task carry lessons that a capable system should not have to relearn. An effective memory system thus lets a frozen model personalize to every user, reason over large scientific corpora, and improve an agent's behavior across long interactions, all without updating its parameters.

Yet the information a task or query needs is rarely known in advance or confined to a single document. A question about energy-boosting foods may hinge on an iron deficiency the user mentioned months earlier in unrelated words, and verifying a claim may require combining facts from several documents.

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.

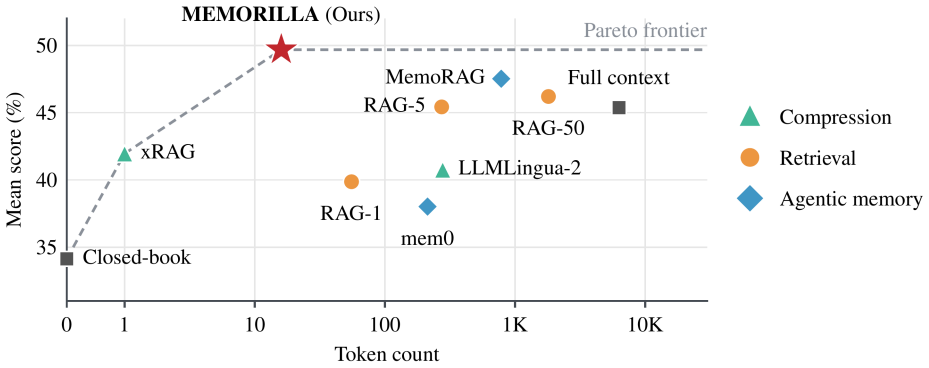


Figure 1: **Performance versus context length.** Mean score over eight benchmarks against the geometric mean of document tokens given to a frozen Qwen3-8B decoder. MEMORILLA lies on the Pareto frontier (dashed) with the highest score at 16 memory tokens, 394× fewer than full context.

Placing the entire collection in the context is costly, and models often fail to locate and combine relevant information even when all of it fits (Hsieh et al., 2024; An et al., 2025; Du et al., 2025; Maharana et al., 2024). Retrieving the text most similar to the query (Lewis et al., 2020; Gao et al., 2023) keeps the context short but misses evidence that resembles the query only weakly, while retrieving more text can yield diminishing returns (Ram et al., 2023). A memory system must therefore extract and combine what each query needs from the entire collection, rather than relying entirely on placing the collection or a subset of its documents in the context as text.

Existing memory systems meet this requirement only in part. Agentic memory systems such as MemGPT (Packer et al., 2023), Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) organize past interactions into summaries or structured records, and MemoRAG (Qian et al., 2025) uses a compressed global memory to guide retrieval, but each ultimately places selected text in the context and inherits the limits of selection. Latent compression methods such as ICAE (Ge et al., 2024), IC-Former (Wang et al., 2024a), AutoCompressors (Chevalier et al., 2023) and ComMer (Zeldes et al., 2025) represent documents as a few soft tokens in the LLM embedding space, but they form these tokens without the query, so the same tokens must serve every question. xRAG (Cheng et al., 2024) incorporates query relevance but only by compressing a single retrieved document. Methods that store model-specific hidden or key–value states, such as LongMem (Wang et al., 2023), MemoryLLM (Wang et al., 2024b), M+ (Wang et al., 2025), and Cartridges (Eyuboglu et al., 2025), tie the memory to one model and require training the LLM, a side network, or a separate cache for each corpus. Overall, none of these approaches reads an entire, growing collection conditioned on each query while supplying a frozen LLM with only a fixed number of tokens (see also Appendix A).

Present work. Here we introduce MEMORILLA, a memory system that learns to retrieve query-relevant information from document embeddings directly into the input embedding space of a frozen LLM (Figure 2). A frozen embedding model encodes each document, such as a user conversation or a book chapter, once into a persistent store. For each query, a trainable memory module attends to every stored embedding conditioned on the query and returns $K{=}16$ memory tokens to the LLM, regardless of collection size. Because the store uses the same embeddings as a dense retrieval index, encoding adds no cost over retrieval, and with 512 documents on one H100, MEMORILLA has 27× lower end-to-end latency than full context answer generation (0.74s against 19.9s, Appendix F).

Our benchmarks mirror three settings in which deployed LLMs need memory (Figures 1 and 3). In personalization, an assistant must adapt to preferences a user rarely states directly, whether answering questions that depend on traits revealed across past conversations (PersonaMem-v2, PersonalizationV4) or writing in the style of the user's earlier text (LaMP-4, LaMP-7). Because a user's preferences rarely share wording with the query, similarity-based retrieval struggles to find and combine relevant information. Notably, on LaMP-4, MEMORILLA reaches 18.15 ROUGE-L against 14.21 for full context while using 474× fewer document tokens. In knowledge-intensive question answering, such as checking a claim against a knowledge base, MEMORILLA outperforms full context on FactKG (84.33% against 80.43%) with 108× fewer document tokens. In long-document understanding, the answer is buried in thousands of tokens of text and often phrased differently from

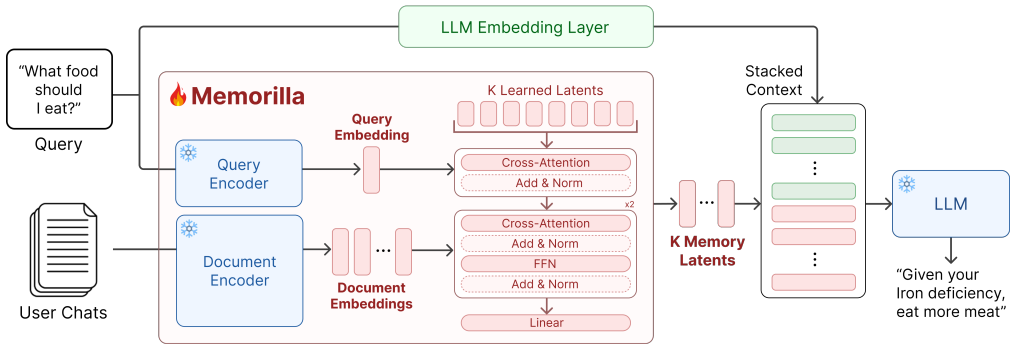


Figure 2: **MEMORILLA architecture.** A frozen encoder embeds source documents (User Chats) once into a persistent store. For each Query, K query-conditioned slots attend over the store and are projected into the frozen LLM's embedding space as memory tokens.

the question, so a system must find and connect relevant content across the entire document. On TriviaQA, whose questions were written independently of their source pages, MEMORILLA reaches the highest exact match (57.00) with 743× fewer document tokens than full context.

Beyond answering questions, MEMORILLA allows agents to learn continually from their own experience. In this work, we consider the TextWorld environment (Côté et al., 2018) where an agent must explore rooms, find and use objects, and complete a multi-step goal within 50 turns, which requires remembering where it has been and what it has already tried. After every turn, MEMORILLA adds that turn to its document store and draws on this experience at the next turn. With our approach, the agent completes 37.5% of held-out games in at least one of three attempts, against 25.0% for the same agent given its full history in context. Overall, MEMORILLA improves the agent's ability to escape navigation loops and execute action sequences that are crucial to reaching the target.

2 MEMORILLA

A persistent latent memory must preserve a document collection across queries while recalling different information for each query. MEMORILLA meets these requirements through three operations: *encoding*, *storage*, and *retrieval*. Given N document chunks $\mathcal{D} = \{d_1, \ldots, d_N\}$, a frozen encoder produces a query-independent representation that is stored once. For instance, $\mathcal{D}$ can be a collection of N user-LLM conversations, chapters of a lengthy book, or health records. For each query q, a trainable memory module reads this store and returns K query-conditioned soft memory tokens, which a frozen LLM consumes together with the query text. Figure 2 summarizes the architecture.

2.1 ENCODING AND PERSISTENT STORAGE

A frozen document encoder f^d_{enc} maps each chunk d_i to an embedding $\mathbf{z}_i \in \mathbb{R}^{h_e}$. Stacking these embeddings produces the query-independent document store

$$\mathbf{Z}_{\mathcal{D}} = [f^d_{\text{enc}}(d_1), \ldots, f^d_{\text{enc}}(d_N)] \in \mathbb{R}^{N \times h_e}. \tag{1}$$

Because $\mathbf{Z}_{\mathcal{D}}$ does not depend on a query, it is computed once and persisted in memory or on secondary storage. Every subsequent query reuses the same matrix, and only added or changed documents must be re-encoded. A frozen query encoder f^q_{enc} then maps q to $\mathbf{z}_q = f^q_{\text{enc}}(q) \in \mathbb{R}^{h_e}$. Unlike $\mathbf{Z}_{\mathcal{D}}$, $\mathbf{z}_q$ is computed only for the current query and is not stored as long-term memory.

2.2 QUERY-CONDITIONED RETRIEVAL (f_{MEM})

In the next phase, a trainable memory module f_{mem} uses $\mathbf{z}_q$ to select information from $\mathbf{Z}_{\mathcal{D}}$:

$$f_{\text{mem}} : (\mathbf{Z}_{\mathcal{D}}, \mathbf{z}_q) \mapsto \mathbf{M}, \qquad \mathbf{M} \in \mathbb{R}^{K \times h_L}, \tag{2}$$

where h_L is the input-embedding dimension of the frozen LLM and $\mathbf{M} = [\mathbf{m}_1; \ldots; \mathbf{m}_K]$ is the output memory matrix. Each row $\mathbf{m}_k \in \mathbb{R}^{h_L}$ is one soft memory token in the LLM embedding space.

The module reads all N stored document embeddings but always contributes exactly K memory tokens to the LLM context, and allows significant context compression (over $4000\times$, see Table 11).

Stage 1: Memory slot initialization. The module initializes K memory slots with learned vectors $\mathbf{Q}_0 \in \mathbb{R}^{K \times h_e}$ that are shared across collections and queries. Distinct initial values define a fixed output budget while allowing the slots to capture different information.

Stage 2: Query conditioning. The slots attend to the query embedding:

$$\mathbf{Q}_c = \text{LayerNorm}(\mathbf{Q}_0 + \text{CrossAttn}(Q{=}\mathbf{Q}_0, K{=}\mathbf{z}_q, V{=}\mathbf{z}_q)) \,. \tag{3}$$

Because the query is represented by one vector, it provides a shared update to all slots; the residual connection preserves the differences among $\mathbf{Q}_0$. The resulting $\mathbf{Q}_c$ is therefore query-conditioned while retaining distinct slot identities.

Stage 3: Document retrieval. The conditioned slots attend to every document embedding through L_{ref} cross-attention blocks, following the latent-array design of Perceiver (Jaegle et al., 2021). Let $\overline{\mathbf{Z}}_{\mathcal{D}} = \text{LayerNorm}(\mathbf{Z}_{\mathcal{D}})$ and $\mathbf{R}_0 = \mathbf{Q}_c$. For block ℓ,

$$\begin{aligned} \widetilde{\mathbf{R}}_\ell &= \mathbf{R}_{\ell-1} + \text{CrossAttn}\big(Q{=}\text{LayerNorm}(\mathbf{R}_{\ell-1}), K{=}\overline{\mathbf{Z}}_{\mathcal{D}}, V{=}\overline{\mathbf{Z}}_{\mathcal{D}}\big) \,, \\ \mathbf{R}_\ell &= \widetilde{\mathbf{R}}_\ell + \text{FFN}\Big(\text{LayerNorm}(\widetilde{\mathbf{R}}_\ell)\Big) \,, \end{aligned} \tag{4}$$

where FFN is a two-layer feed-forward network with GELU activations (Hendrycks & Gimpel, 2023). Each slot can attend to the full collection, while the query-conditioned slots determine which information is retained. The refinement blocks therefore implement a continuous, query-conditioned read over all stored documents rather than a discrete top-k selection.

Stage 4: Projection to the LLM embedding space. A learned projection maps the refined slots from the encoder dimension h_e to the LLM input-embedding dimension h_L:

$$\mathbf{M} = \text{LayerNorm}(\mathbf{R}_{L_{\text{ref}}})\mathbf{W}_{\text{proj}}, \qquad \mathbf{W}_{\text{proj}} \in \mathbb{R}^{h_e \times h_L}. \tag{5}$$

The resulting rows $\mathbf{m}_1, \ldots, \mathbf{m}_K$ are the soft memory tokens passed to the frozen LLM.

2.3 Memory Injection into the Frozen LLM

Finally, the LLM input-embedding layer maps the text query to $\mathbf{P}_q$, giving

$$\mathbf{X}_{\text{LLM}} = [\underbrace{\mathbf{m}_1, \ldots, \mathbf{m}_K}_{\text{soft memory tokens}}; \mathbf{P}_q]. \tag{6}$$

Because $\mathbf{m}_k \in \mathbb{R}^{h_L}$ has the same dimension as a text token embedding, the LLM processes memory and text in the same input sequence. No LLM layer or attention mechanism is changed. Rotary position embeddings are applied over the combined sequence in the standard way (Su et al., 2021). The frozen LLM can therefore attend to the recalled memory through its existing self-attention layers.

2.4 Supervised Training Objective

The frozen LLM generates $\mathbf{Y}_{\text{out}} = (y_1, \ldots, y_L)$ autoregressively from $\mathbf{X}_{\text{LLM}}$, and we minimize the negative log-likelihood of the target answer tokens:

$$\mathcal{L}_{\text{sup}} = -\sum_{i=1}^{L} \log P(y_i \mid \mathbf{y}_{<i}, \mathbf{X}_{\text{LLM}}; \boldsymbol{\theta}_{\text{mem}}), \tag{7}$$

where $\boldsymbol{\theta}_{\text{mem}}$ includes the memory module and its projection to the LLM embedding space. The loss is applied only to answer positions, not to the prompt or memory positions.

2.5 Memorilla with Qwen3

Based on the formalization above, we instantiate Memorilla with f^d_{enc} and f^q_{enc} as a frozen Qwen3-Embedding-4B model. Unless stated otherwise, we fix $K = 16$ throughout the paper and use a Qwen3-8B model as the frozen LLM. This gives us 194,123,776 trainable parameters.

2.6 Training Stages

We use a three-stage curriculum that moves from general content preservation to diverse language tasks and then to the evaluation domains. See Appendix B.2 for training hyperparameters.

Stage 1 trains the memory module on 1.98 million English Wikipedia paraphrase reconstruction examples. Each passage is divided into document chunks, and the module must preserve enough information in its fixed memory budget for the frozen LLM to generate a paraphrase. This stage provides a general initialization before the module encounters task-specific supervision.

Stage 2 continues training on 1.34 million examples drawn from ten datasets covering reading comprehension, open-domain question answering, and summarization. The variation in source structure and answer format teaches the module to recall different information from the same type of stored representation. Table 1 lists the component datasets and their sizes.

Stage 3 fine-tunes the resulting module on a unified mixture of the training splits from all eight evaluation benchmarks. One multi-task checkpoint is used directly for FactKG, TriviaQA, NarrativeQA, PubMedQA, LaMP-4, and LaMP-7. PersonaMem-v2 and PersonalizationV4 receive additional benchmark-specific fine-tuning because their validation losses required more epochs to plateau, while the other benchmarks plateaued or began to overfit earlier.

3 Experiments

3.1 Baselines

We compare Memorilla against RAG approaches based on cosine similarity, Mem0 (Chhikara et al., 2025), and MemoRAG (Qian et al., 2025). xRAG (Cheng et al., 2024) and ComMer (Zeldes et al., 2025) (via its reported results, Table 12) compress the context into latent input embeddings for the LLM. Finally, we also compare against LLMLingua-2 (Pan et al., 2024), which is a task-agnostic text based prompt compression method. All baselines we run use a frozen Qwen3-Embedding-4B encoder for document embeddings and a frozen Qwen3-8B decoder for generation. Mem0 and MemoRAG additionally use a frozen Qwen3-4B to build their memories, matching the 4B scale of the encoder with which Memorilla reads the same collections (Appendix C).

3.2 Benchmarks

We evaluate Memorilla on eight supervised benchmarks spanning (a) personalization, (b) long-document QA, and (c) knowledge-intensive QA tasks that demand memory. For personalization, we consider PersonaMem-v2 (PMv2) (Jiang et al., 2025), LaMP-4, LaMP-7 (Salemi et al., 2023) and our own PersonalizationV4 (PV4) benchmarks as they require integrating information scattered across a user's prior chats or writing history to form a coherent user representation. For long-document QA, we choose NarrativeQA (Kočiský et al., 2017) and TriviaQA (Joshi et al., 2017) to test the method's ability to compress long documents in their entirety into a usable representation for downstream QA. Finally, for knowledge-intensive QA, we consider FactKG (Kim et al., 2023) and PubMedQA (Jin et al., 2019) to test factual grounding from structured or semi-structured evidence of varying quantity and format. Table 10 (Appendix D.2) reports how many document tokens each method gives the LLM per query across all the benchmarks.

3.3 Evaluation

We use accuracy for classification tasks, exact match and token-level F1 (lowercased, whitespace-tokenized) for short-form QA, and ROUGE, BLEU, F1, and METEOR for open-ended text generation. PMv2 and PV4 are evaluated on their short-answer questions. The model's generation is embedded with the frozen encoder and scored as correct if the gold answer is its cosine nearest neighbor among the question's predefined answer choices, which we report as short-answer accuracy. All supervised LLM evaluations use a decoding temperature of 0 to ensure reproducibility. Our serving stack enforces a 32,768-token decoder context and enables YaRN whenever an assembled full-context input exceeds this limit (Appendix D).

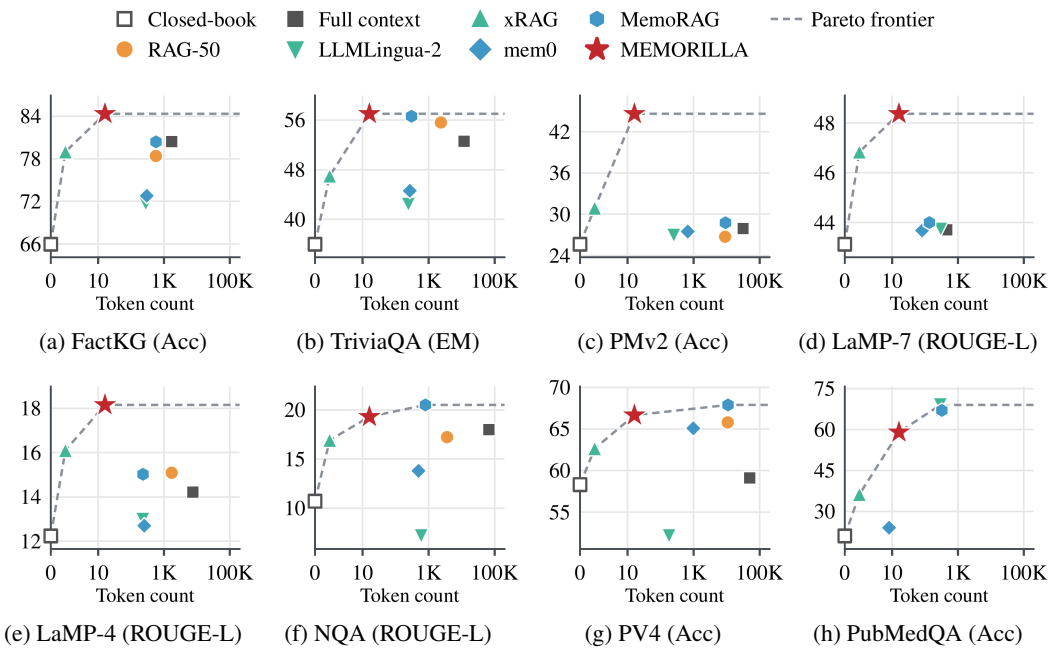


Figure 3: **Per-benchmark performance versus context length.** Primary-metric score against mean document tokens per query, with a frozen Qwen3-8B decoder. MEMORILLA lies on the Pareto frontier (dashed) of every benchmark, leads on five and is second on NarrativeQA and PV4.

4 QUANTITATIVE RESULTS

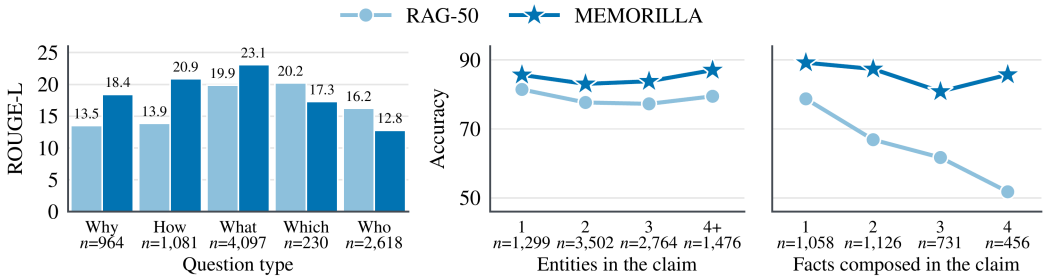
MEMORILLA outperforms all baselines on average while using 394×, 113× and 49× fewer document tokens than full context, RAG-50 and MemoRAG in geometric mean (Figure 1). It also lies on the Pareto frontier of every benchmark, while the compact baselines xRAG, LLMLingua-2 and Mem0 average only 41.92, 40.75 and 38.02 against 49.68. Figure 3 breaks this down by benchmark, where MEMORILLA leads on five and and is second on NarrativeQA and PV4.

Relational evidence. On FactKG (Figure 3a), MEMORILLA reaches 84.33% against 80.43% for full context, 80.38% for MemoRAG and 78.98% for xRAG, with 108× and 36× fewer tokens than the first two. Because full context receives every document, the margin cannot stem from discarded evidence, and Section 5 shows that MEMORILLA's margin over RAG-50 grows with the number of entities and widens overall with the number of composed facts.

Long-document and personalization QA. On TriviaQA (Figure 3b), MEMORILLA reaches 57.00 EM against 56.60 for MemoRAG and 52.60 for full context with 19× and 743× fewer document tokens. On PMv2 short-answer (Figure 3c), MEMORILLA reaches 44.60% against 30.90% for xRAG, the strongest baseline. Every other baseline stays within about three points of closed-book (25.64%), including full context, which uses 1,974× more document tokens than MEMORILLA. On LaMP-7, every baseline except xRAG (46.84) stays within one ROUGE-L point of the closed-book model, whereas MEMORILLA improves it from 43.13 to 48.37. Finally, MEMORILLA achieves 18.15 ROUGE-L on LaMP-4 (Figure 3e) with 474× fewer tokens than the full context baseline.

Where memory does not help. On the remaining three benchmarks (Figures 3f–3h), a baseline leads, but only by supplying the LLM with 18 to 690 times more document tokens. MemoRAG leads NarrativeQA in ROUGE-1 and ROUGE-L (21.80 and 20.51 versus 19.68 and 19.31) with 50× more document tokens, consistent with its access to verbatim passages, and PV4 (67.88% versus 66.65%) with 690× more. On PubMedQA, whose four abstracts total only 324 tokens, LLMLingua-2 reaches 69.00% and four other baselines 67.00%, versus 59.00% for MEMORILLA.

Full context is not an upper bound. Full context trails MEMORILLA on every benchmark except PubMedQA and on 20 of 21 metrics in Table 11, and RAG-50 exceeds it on 15, so supplying the entire collection hurts more often than it helps.



(a) NarrativeQA by question type (b) FactKG by entity count (c) FactKG by composition

Figure 4: **Per-question breakdowns against RAG-50. (a)** NarrativeQA question types are the question's first word. Conceptual questions ('Why', 'How', 'What') favor MEMORILLA whereas entity-recall questions ('Which', 'Who') favor RAG. **(b)** MEMORILLA's advantage grows with entity count, indicating better capture of multi-entity relational patterns. **(c)** RAG-50 degrades as more facts are composed, while MEMORILLA remains stable across all depths.

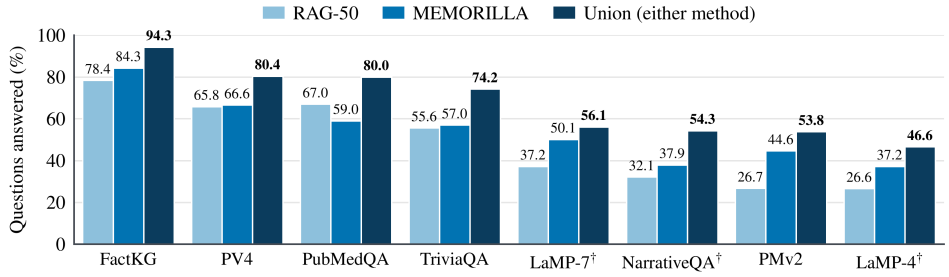


Figure 5: **Per-question complementarity between MEMORILLA and RAG-50.** Share of questions answered by each method alone and by at least one of them (union), an oracle upper bound for a hybrid system. The union exceeds either method on every benchmark. †A question counts as answered when its ROUGE-L is at least $\max(0.20, m)$, where m is MEMORILLA's median.

5 WHAT DOES MEMORILLA LEARN?

Conceptual understanding vs. factual recall. Figure 4a breaks down NarrativeQA performance by question type. MEMORILLA outperforms RAG-50 by +4.91 ROUGE-L on "why" questions and +7.02 on "how," both requiring reasoning about narrative structure and causality. Conversely, RAG-50 leads by +3.45 on "who" and by +2.93 on "which," where retrieval can locate the entity name verbatim. One explanation is that compression preserves global structure for conceptual reasoning but loses the exact surface tokens that retrieval excels at pinpointing (although question type also varies with answer length and entity frequency). The same trade-off would explain why MemoRAG, which reads selected passages, leads our ROUGE scores on this benchmark while trailing on F1. Figure 6 illustrates the complementary strength in personalization, where MEMORILLA uses semantically relevant medical history despite no lexical overlap with the query.

Multi-entity relational reasoning. On FactKG, MEMORILLA's advantage over RAG-50 grows monotonically with entity count, from +4.23pp at 1 entity to +7.59pp at 4+ (Figure 4b). RAG-50 drops from 78.73 to 51.75 as the number of composed facts increases from 1 to 4, while MEMORILLA remains in the 80–89 range (Figure 4c). Compression encodes relational structure holistically, whereas retrieval degrades when the reasoning chain spans multiple documents. Because 73.5% of FactKG claims have more than 50 chunks, part of this gap reflects evidence RAG-50 never retrieves, but full context, which reads every chunk, still trails MEMORILLA overall (80.43 against 84.33).

MEMORILLA and RAG solve different questions. On FactKG, we observed that 15.9% of questions are answered by MEMORILLA alone against 10.0% by RAG-50 alone, and on PMv2 short-answer the split is 27.1% against 9.2%. In Figure 5, we show that the union of MEMORILLA and RAG (which is an oracle upper bound), could lead to substantial gains in performance than either approach alone, which motivates the design of hybrid retrieval and latent-memory systems as potential future work.

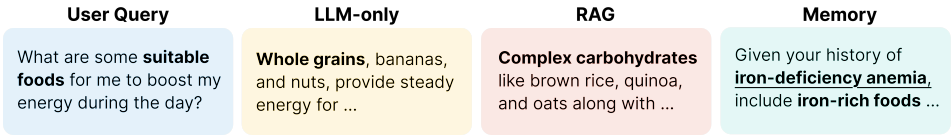


Figure 6: **Personalization Case Study.** On PersonalizationV4, MEMORILLA personalizes a generic query by leveraging the user's remembered iron-deficiency anemia history, despite no lexical overlap between the question and the relevant conversation history.

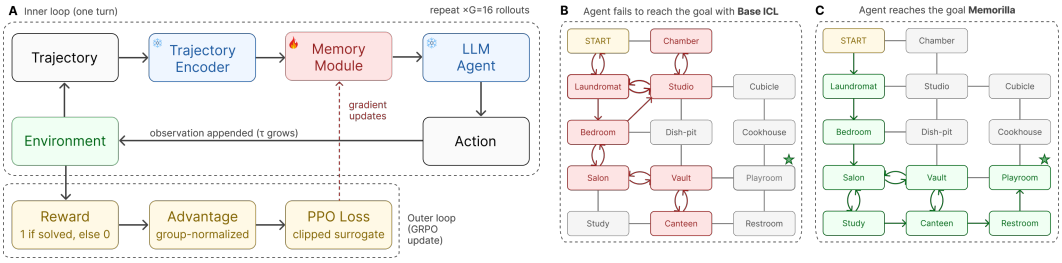


Figure 7: **Reinforcement-learned memory on TextWorld.** **(a)** A frozen encoder embeds the trajectory, the trainable memory module compresses it to K=8 latents, and the frozen LLM agent acts; sparse rewards over G=16 rollouts drive a GRPO update to the memory module alone. **(b)** On a held-out game, ICL loops between explored rooms while **(c)** MEMORILLA reaches the goal ($\star$).

What neither method solves. We also examine the questions that both MEMORILLA and RAG-50 answer incorrectly. On PV4, many of these require linking facts that each appear somewhere in the user's history, which suggests that the remaining errors come from combining information rather than from failing to find it. On FactKG, both methods are biased toward judging claims false. True claims make up 89.5% of the claims both methods get wrong but only 28.7% of those both get right, which points to a miscalibrated decision threshold rather than missing evidence.

6 CONTINUAL LEARNING WITH MEMORILLA

Beyond the supervised learning settings where the document collection can be static, we showcase the potential of MEMORILLA in scenarios where an LLM agent has to continuously learn and adapt from its environment. One such environment is TextWorld (Côté et al., 2018), a suite of partially observable text games in which an agent explores up to 65 rooms, manipulates objects and pursues multi-step goals for at most 50 turns, with reward arriving mainly on a win. After each turn, MEMORILLA stores the completed turn as a new document and condenses the growing document embedding store into $K = 8$ memory tokens placed before the prompt (Figure 7 and Appendix G).

6.1 EXPERIMENTAL SETUP

The agent is a frozen Qwen3-4B decoder paired with the frozen Qwen3-Embedding-4B encoder. Only the memory module is trained, from random initialization and with GRPO on environment reward (see Appendix G.1). We compare against two baselines. Full-history in-context learning (ICL) gives the decoder the complete interaction history without memory, and an untrained memory module isolates the contribution of RL. Unlike in the supervised setting, the memory supplements the text rather than replacing it. Every condition keeps the full history in a 16,384-token window, so any improvement reflects what the memory learns to extract, not access to additional information. In the medium and hard games, the history grows from 800 tokens at the first turn to about 3,200 and then levels off, since the median rollout ends at turn 33 and only 5 of 40 rollouts reach turn 50.

6.2 RESULTS

Task completion. Training MEMORILLA with GRPO improves avg@3 from 0.186 to 0.242 and pass@3 from 0.250 to 0.375 relative to full-history ICL (Figure 8). Untrained memory instead falls

below ICL on both metrics, avg@3 (0.153) and pass@3 (0.225), so the gain comes from learning rather than from the added tokens. The trained memory acts as a compact state that keeps the frozen agent oriented over long trajectories.

MEMORILLA reduces agent loops. The largest behavioral change is termination rather than solve rate. ICL times out on 71% of rollouts against 58% for MEMORILLA despite both reading the same trajectory text. Trajectory inspection shows that trained memory helps the agent escape navigation loops and execute keyed action sequences, such as unlocking and then opening a container. Figure 7 illustrates this, with ICL cycling between explored rooms and never reaching the goal, while MEMORILLA advances to the target room. The remaining failures concentrate in the longest games, whose quests approach or exceed the 50-turn limit and require backtracking across many rooms.

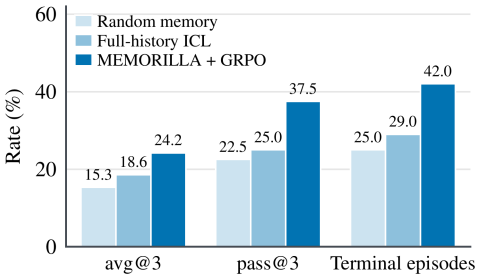


Figure 8: **MEMORILLA on TextWorld** with a frozen Qwen3-4B decoder and a 16K context. Terminal episodes end before the turn limit.

7 DISCUSSION AND CONCLUSION

MEMORILLA successfully integrates learned memory into the input space of frozen LLMs, addressing the fundamental limitation that current systems treat accumulated experience as text to be fetched rather than knowledge to be learned. A single trainable module compresses hundreds of documents into 16 latent tokens, leading retrieval, compression and agent memory baselines on average at 20 to 4,212$\times$ fewer document tokens across personalization, long-document QA, and knowledge-intensive tasks. The same interface can also be optimized by RL, where training only the memory module in TextWorld improves pass@3 over full-history in-context learning and substantially reduces non-terminating rollouts for a frozen 4B agent. The injection mechanism is shared between the two settings, so the memory module can learn from answer tokens or from sparse reward, though it supplements the trajectory for the agent rather than compressing it. More broadly, a document store written once and read differently for every query sits between retrieval and long-context decoding, personalizing to users without placing their histories in the prompt. Together with the TextWorld results, this suggests memory modules could become reusable components shared across applications.

Limitations. MEMORILLA can lose fine-grained factual details, likely because the document encoder was never trained for memory compression, and training a dedicated encoder that preserves these details is a natural next step. The memory module must also be trained separately for each LLM's embedding space, which rules out closed-source models, and because every result pairs a Qwen encoder with a Qwen decoder, we have not tested how well it transfers across model families. Our evaluation covers personalization, long-document QA and knowledge-intensive QA, so MEMORILLA may not carry over directly to domains such as coding or scientific questions outside biomedicine. Finally, because decoding dominates latency, the efficiency gains show up mainly for large collections.

Future work. Our results open several promising directions. Pairing latent memory with text retrieval could combine the global structure that memory captures with the exact entities that retrieval recovers, and encoders trained specifically for compression could close the remaining gap on fine-grained details. Because the store holds only embeddings, a multimodal encoder could extend the same interface to images, audio and video, giving assistants a memory of what a user has seen and heard. Since each user's memory is a separate store, it can be kept on device or deleted on request, which eases privacy concerns. Finally, portable memory modules could let a single store serve many models.

Conclusion. We introduced MEMORILLA, a latent semantic memory that allows frozen LLMs to learn from accumulated experience instead of retrieving it as text. Across personalization, long-document QA and knowledge-intensive QA, it outperforms retrieval, compression and agent memory baselines with a fraction of the context. It also enables continual learning in agentic RL, where training only the memory module teaches an agent a new environment without overwriting prior knowledge. We hope this work paves the way toward hybrid systems that pair latent memory with text retrieval, allowing LLMs to accumulate and use knowledge over long-context tasks.

AI USE STATEMENT

We used coding agents to implement and run experiments, and minimally used generative AI tools to polish the writing of the paper. As part of our method, GPT-5 generated the PersonalizationV4 personas, GPT-5.2 generated the question categories, chat topics and questions, and GPT-5-mini generated the dialogues (Appendix D.1). We did not use generative AI tools for any other part of this work. The authors reviewed all AI-assisted work and verified the code and experimental results produced with coding agents before reporting them. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

MEMORILLA stores and recalls information from user conversations, which raises privacy concerns when deployed on real user data. Because the memory store is external to the LLM, a user's embeddings can be removed without retraining, but stored embeddings can still encode sensitive information and deployments should obtain user consent and protect them accordingly. PersonalizationV4 consists of synthetic users based on LLM-generated personas and contains no data from real users. All 1,206 persona files follow a single machine template and were written in one batch on January 29, 2026. All other benchmarks are publicly available.

REPRODUCIBILITY STATEMENT

Section 2 describes the MEMORILLA architecture, and Sections 2.6 and 2.4 describe the training curriculum and objective. Appendix B.1 lists the training data, and Appendix B.2 reports the model configuration, training hyperparameters, document chunking and evaluation splits. Appendix C describes the implementation of each baseline, and Appendix D describes the benchmarks, the PersonalizationV4 data pipeline and the context budget of each method. Appendix G gives the full TextWorld RL setup. All supervised evaluations use greedy decoding and a fixed random seed. We will release the code, trained checkpoints and the PersonalizationV4 data upon publication.

REFERENCES

Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why does the effective context length of LLMs fall short? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eoln5WgrPx.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset. 2016. doi: 10.48550/arXiv.1611.09268. URL https://arxiv.org/abs/1611.09268.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.

Yinpeng Chen, DeLesley Hutchins, Aren Jansen, Andrey Zhmoginov, David Racz, and Jesper Sparre Andersen. MELODI: Exploring memory compression for long contexts. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=TvGPP8i18S.

Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. Dialogsum: A real-life scenario dialogue summarization dataset. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 5062–5074. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-acl.449. URL https://aclanthology.org/2021.findings-acl.449.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xrag: Extreme context compression for retrieval-augmented generation with one token. *Advances in Neural Information Processing Systems*, 37:109487–109516, 2024.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. 2025. doi: 10.48550/arXiv.2504.19413. URL https://arxiv.org/abs/2504.19413.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. In *Computer Games Workshop at IJCAI*, 2018. URL https://arxiv.org/abs/1806.11532.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts llm performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 2025.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. 2025. doi: 10.48550/arXiv.2506.06266. URL https://arxiv.org/abs/2506.06266.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hB0J8MqqXV.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-5409. URL https://aclanthology.org/D19-5409.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. 2020. doi: 10.48550/arXiv.2002.08909. URL https://arxiv.org/abs/2002.08909.

Jie He, Richard He Bai, Sinead Williamson, Jeff Z Pan, Navdeep Jaitly, and Yizhe Zhang. Clara: Bridging retrieval and generation with continuous latent reasoning. *arXiv preprint arXiv:2511.18659*, 2025.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023. URL https://arxiv.org/abs/1606.08415.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

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

Bernd Huber, Ghazal Fazelnia, Andreas Damianou, Sebastian Peleato, Maksym Lefarov, Praveen Chandar, Marco De Nadai, Mounia Lalmas, and Paul N. Bennett. Embedding-to-prefix: Continual personalization with large language models. In *AI That Keeps Up: NeurIPS 2025 Workshop on Continual and Compatible Foundation Model Updates*, 2025.

Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. 2021. doi: 10.48550/arXiv.2103.03206. URL https://arxiv.org/abs/2103.03206.

Pegah Jandaghi, XiangHai Sheng, Xinyi Bai, Jay Pujara, and Hakim Sidahmed. Faithful persona-based conversational dataset generation with large language models, 2023. URL https://arxiv.org/abs/2312.10007.

Bowen Jiang, Yuan Yuan, Maohao Shen, Zhuoqun Hao, Zhangchen Xu, Zichen Chen, Ziyi Liu, Anvesh Rao Vijjini, Jiashu He, Hanchao Yu, Radha Poovendran, Gregory Wornell, Lyle Ungar, Dan Roth, Sihao Chen, and Camillo Jose Taylor. Personamem-v2: Towards personalized intelligence via learning implicit user personas and agentic memory. 2025. doi: 10.48550/arXiv.2512.06688. URL https://arxiv.org/abs/2512.06688.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering, 2019. URL https://arxiv.org/abs/1909.06146.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. 2017. doi: 10.48550/arXiv.1705.03551. URL https://arxiv.org/abs/1705.03551.

Jikun Kang, Wenqi Wu, Filippos Christianos, Alex J Chan, Fraser Greenlee, George Thomas, Marvin Purtorab, and Andy Toulis. Lm2: Large memory models. *arXiv preprint arXiv:2502.06049*, 2025.

Jiho Kim, Sungjin Park, Yeonsu Kwon, Yohan Jo, James Thorne, and Edward Choi. Factkg: Fact verification via reasoning on knowledge graphs. 2023. doi: 10.48550/arXiv.2305.06590. URL https://arxiv.org/abs/2305.06590.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. 2017. doi: 10.48550/arXiv.1712.07040. URL https://arxiv.org/abs/1712.07040.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2020. doi: 10.48550/arXiv.2005.11401. URL https://arxiv.org/abs/2005.11401.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models, 2023. URL https://arxiv.org/abs/2301.12597.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25081–25091, 2025.

Maxime Louis, Hervé Déjean, and Stéphane Clinchant. Pisco: Pretty simple compression for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15506–15521, 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.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Lin Ning, Luyang Liu, Jiaxing Wu, Neo Wu, Devora Berlowitz, Sushant Prakash, Bradley Green, Shawn O'Banion, and Jun Xie. User-llm: Efficient llm contextualization with user embeddings. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 1219–1223, 2025.

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.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression, 2024. URL https://arxiv.org/abs/2403.12968.

Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*, pp. 2366–2377, 2025.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 784–789. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

David Rau, Shuai Wang, Hervé Déjean, and Stéphane Clinchant. Context embeddings for efficient answer generation in rag. *arXiv preprint arXiv:2407.09252*, 2024.

Siva Reddy, Danqi Chen, and Christopher D. Manning. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://direct.mit.edu/tacl/article/43511.

Anna Rogers, Olga Kovaleva, Matthew Downey, and Anna Rumshisky. Getting closer to ai complete question answering: A set of prerequisite real tasks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8722–8731, 2020. doi: 10.1609/aaai.v34i05.6398. URL https://ojs.aaai.org/index.php/AAAI/article/view/6398.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization. 2023. doi: 10.48550/arXiv.2304.11406. URL https://arxiv.org/abs/2304.11406.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1073–1083. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1099. URL https://aclanthology.org/P17-1099.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. (arXiv:2303.11366), October 2023. doi: 10.48550/arXiv.2303.11366. URL http://arxiv.org/abs/2303.11366. arXiv:2303.11366.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. 2021. doi: 10.48550/arXiv.2104.09864. URL https://arxiv.org/abs/2104.09864.

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.

Xiangfeng Wang, Zaiyi Chen, Tong Xu, Zheyong Xie, Yongyi He, and Enhong Chen. In-context former: Lightning-fast compressing context for large language model. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2445–2460, 2024a.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, et al. Memoryllm: Towards self-updatable large language models. *arXiv preprint arXiv:2402.04624*, 2024b.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending memoryLLM with scalable long-term memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=OcqbkROe8J.

Yaxiong Wu, Sheng Liang, Chen Zhang, Yichao Wang, Yongyue Zhang, Huifeng Guo, Ruiming Tang, and Yong Liu. From human memory to ai memory: A survey on memory mechanisms in the era of llms. *arXiv preprint arXiv:2504.15965*, 2025.

Zijun Wu, Yongchang Hao, and Lili Mou. Tokmem: One-token procedural memory for large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RWjEf9PdiJ.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. (arXiv:2210.03629), March 2023. doi: 10.48550/arXiv.2210.03629. URL http://arxiv.org/abs/2210.03629. arXiv:2210.03629.

Yoel Zeldes, Amir Zait, Ilia Labzovsky, Danny Karmon, and Efrat Farkash. Commer: a framework for compressing and merging user data for personalization. *arXiv preprint arXiv:2501.03276*, 2025.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=1eQT9OzfNQ.

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

A Related Work

Textual memory. Textual memory keeps source records readable and retrieves relevant records at inference. RAG (Lewis et al., 2020) and REALM (Guu et al., 2020) retrieve passages from a document index. Systems built for longer histories retain the same basic interface while organizing the records differently. MemGPT (Packer et al., 2023) moves information between context tiers, MemoryBank (Zhong et al., 2024) stores conversations and summaries, and Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) maintain structured records across sessions. These systems preserve provenance and exact details, but the LLM can reason only over material that retrieval places in its context. Evidence outside the selected set remains unavailable, and widening that set yields limited returns. In-Context RALM (Ram et al., 2023) finds that most question-answering gains come from the first one or two retrieved documents. MemoRAG (Qian et al., 2025) uses compressed global memory to generate retrieval clues, but its final generator still reads selected passages. Textual memory is therefore effective when evidence can be localized, while signals distributed across a collection remain constrained by document selection. MEMORILLA instead computes a query-conditioned latent state from the full collection, complementing retrieval when exact evidence is needed.

Latent compression. Latent compression replaces replayed text with compact continuous state. ICAE (Ge et al., 2024) and IC-Former (Wang et al., 2024a) encode a long input into reusable vectors consumed by frozen target LLMs, while AutoCompressors (Chevalier et al., 2023) fine-tune an LLM to produce and consume recursive summary vectors. These representations can be cached because they are computed without a query, but they cannot change what they emphasize for different queries. Gist tokens (Mu et al., 2023) and 500xCompressor (Li et al., 2025) make the same tradeoff using layer-specific key–value states rather than standard input embeddings. xRAG (Cheng et al., 2024) makes the representation query relevant by first selecting a document through retrieval and mapping its embedding to one LLM token, which retains document selection as the access mechanism. A second line improves the alignment between compression and generation. PISCO (Louis et al., 2025) and COCOM (Rau et al., 2024) jointly adapt their compressors and generators, while CLaRa (He et al., 2025) learns reusable document vectors before training a query reasoner with the generator. These methods gain a learned latent interface but require an adapted generator. Other methods obtain reuse within a narrower memory domain. TokMem (Wu et al., 2026) optimizes a separate token for each procedure, while ComMer (Zeldes et al., 2025), USER-LLM (Ning et al., 2025), and Embedding-to-Prefix (Huber et al., 2025) construct latent state for user histories. MEMORILLA instead accepts unseen document collections without further optimization and converts an LLM-independent store into new query-conditioned input embeddings for a frozen LLM.

Model memory. Model memory stores information in architecture-specific state. MemoryLLM (Wang et al., 2024b) maintains a self-updating latent pool, M+ (Wang et al., 2025) retrieves from long-term storage using the current hidden state, and LongMem (Wang et al., 2023) retrieves cached key–value states through a trained side network. Their shared internal representation supports query-conditioned recall, but also couples the memory and its reader to a particular model. Other methods preserve state during model execution rather than across an independent document collection. LM2 (Kang et al., 2025) updates a gated memory matrix during autoregressive inference. Titans (Behrouz et al., 2025) combines fixed task-level parameters with a contextual neural memory updated while processing a sequence. MELODI (Chen et al., 2025) and Activation Beacon (Zhang et al., 2025) compress state across context windows and transformer layers to extend long-sequence processing. Cartridges (Eyuboglu et al., 2025) makes model-specific key–value caches reusable and composable, but requires offline optimization for each new corpus. These approaches demonstrate the value of persistent latent state, while their architecture-specific interfaces limit reuse across models and collections. MEMORILLA instead keeps memory external and supplies it to a frozen LLM as input embeddings.

B Additional Details on Training Methodology

B.1 Training Datasets

Table 1 details the training data used across the first two stages of the MEMORILLA pipeline. The pretraining corpus consists of approximately 2M paraphrase reconstruction examples derived from

English Wikipedia articles. The finetuning mixture aggregates 1.34M queries across 10 sources spanning reading comprehension, open-domain QA, and summarization, exposing the memory module to varied compression demands across different answer types and document structures. PubMedQA contributes only its 900-question training split, which is disjoint from the 100 test questions used for evaluation.

Table 1: **Training datasets for MEMORILLA.** The pretraining corpus is derived from English Wikipedia. The finetuning mixture spans reading comprehension, open-domain question answering, and summarization tasks across 10 sources. Sizes are numbers of training queries.

Stage	Source	Category	Num. Queries
Pretraining	EnWiki Articles	Paraphrase	1,979,809
Finetuning	SQuAD v2 (Rajpurkar et al., 2018)	Reading Comprehension	86,821
	DROP (Dua et al., 2019)	Reading Comprehension	77,400
	CoQA (Reddy et al., 2019)	Reading Comprehension	108,647
	PubMedQA (Jin et al., 2019)	Reading Comprehension	900
	QuAIL (Rogers et al., 2020)	Reading Comprehension	10,246
	PwC (Ge et al., 2024)	Reading Comprehension	241,564
	CNN/DailyMail (See et al., 2017)	Summarization	287,113
	SamSum (Gliwa et al., 2019)	Summarization	14,731
	DialogSum (Chen et al., 2021)	Summarization	12,460
	MS MARCO (Bajaj et al., 2016)	Open-domain QA	504,111
	Finetuning total		**1,343,993**

B.2 TRAINING CONFIGURATION

All MEMORILLA runs use frozen encoder and decoder backbones. Only the memory module is trainable unless otherwise noted. Supervised training stages use bf16 mixed precision on 4 NVIDIA H100 80GB GPUs and random seed 23. Document embeddings are precomputed and cached to amortize encoding cost. Table 2 lists how each benchmark's collection is split into the chunks d_i that the encoder embeds. The supervised objective follows Section 2.4, with loss computed only over answer tokens. A single memory module is trained jointly across all downstream tasks for up to 5 epochs. On PV4 and PMv2 we continue for additional epochs because both were still improving on validation when the other benchmarks had converged. Train and test users are disjoint on both, so one module serves every user and enrolling a new one requires only embedding their chats.

Table 2: **Document chunks per benchmark.** Each chunk d_i is embedded once by the frozen encoder. Token counts use the Qwen3 tokenizer.

Benchmark	Chunk unit	Tokens per chunk (mean / median)
PMv2	One conversation session	280 / 262
PV4	One chat dialogue	252 / 224
PubMedQA	One abstract section	84 / 78
TriviaQA	Evidence passage, split sentence-greedily at 128 tokens	70 / 77
NarrativeQA	Story paragraph, split on sentences above $\sim$256 tokens	94 / 35
FactKG	One knowledge-graph triple	13 / 13
LaMP-4	One profile item	47 / 44
LaMP-7	One profile tweet	25 / 25

C ADDITIONAL DETAILS ON BASELINES

All baselines use the frozen Qwen3-8B as the decoder and the frozen Qwen3-Embedding-4B as the document encoder, and every answer is generated greedily. Baselines that insert text share one prompt template per benchmark and differ only in the context placed before the question.

Closed-book. The decoder receives only the query, which shows how much of each benchmark it answers from parametric knowledge.

RAG and Full Context. RAG-k ranks the chunks of a collection by the cosine similarity between their embeddings and the query embedding, and places the top k chunks in the prompt as text, most similar first and separated by blank lines. Full Context places every chunk in its stored order. Inputs

Table 3: **Model configuration for MEMORILLA.** The memory module maps document representations into a fixed set of learned memory tokens that are inserted into the frozen decoder context.

Component	Setting	Value
Memory module	Pooling mode	Adaptive cross-attention with softmax over documents
	Memory slots	$K=16$
	Cross-attention layers	2 layers, slots ← documents
	Attention heads	8
	Encoder-side dimension	2560
	Decoder-side dimension	2560 for 4B decoder, 4096 for 8B decoder
	Slot initialization	Xavier uniform
	Projection	Linear encoder-to-decoder projection
	Regularization	Dropout 0.1 with residual cross-attention
Frozen backbones	Document encoder	Qwen3-Embedding-4B, mean-pooled per document
	4B decoder	Qwen3-4B
	8B decoder	Qwen3-8B
	Trainable parameters	Memory module only; 190,190,080 for 4B and 194,123,776 for 8B
Generation	Decoding	Greedy decoding with $T=0$, top-$p=1.0$, top-$k=-1$
	Short-answer limit	8 tokens for classification, 64 tokens for QA and fact-verification tasks
	Long-form limit	128 tokens for LaMP-4 and LaMP-7; 512 tokens for EnWiki reconstruction

Table 4: **Training stages for MEMORILLA.** Stage 1 pretrains memory on EnWiki paraphrase reconstruction, Stage 2 warm-starts on the mixed supervision corpus in Table 1, and Stage 3 performs downstream multi-task fine-tuning.

Setting	Stage 1 EnWiki pretrain	Stage 2 Multi-task pretrain	Stage 3 Downstream fine-tune
Initialization	Random	Warm-start from Stage 1	Warm-start from Stage 2
Data	EnWiki paraphrase reconstruction	Mixed supervision corpus	Unified downstream mixture
Tasks	Paraphrase reconstruction	Reading comprehension, open-domain QA, summarization	PV4, PMv2, FactKG, TriviaQA, NQA, PubMedQA, LaMP-4, LaMP-7
Optimizer	AdamW	AdamW	AdamW
LR scheduler	Cosine annealing with warmup	Cosine annealing with warmup	Cosine annealing with warmup
Peak learning rate	3×10^{-4}	3×10^{-4}	1×10^{-4}
Weight decay	1×10^{-2}	1×10^{-2}	1×10^{-2}
Warmup ratio	0.03	0.03	0.1
Max decoder length	1024	1024	2048
Per-device batch size	8 for 4B, 4 for 8B	8 for 4B, 4 for 8B	2
Gradient accumulation	4 for 4B, 8 for 8B	4 for 4B, 8 for 8B	8
Effective batch size	128	128	64
Devices	4 GPUs	4 GPUs	4 GPUs
Epochs	1	1	5
Checkpoint selection	Final checkpoint	Final checkpoint	One shared multi-task checkpoint; best by validation loss only for PV4/PMv2 specialization
Validation interval	—	—	0.25 epoch
Random seed	23	23	23

longer than the enforced 32,768-token limit use YaRN with a scaling factor of 4, which extends the window to 131,072 tokens and applies to part of PMv2, TriviaQA and NarrativeQA and to all of PV4 (Table 10). For the 10.8% of evaluated NarrativeQA questions whose books exceed this window, the earliest chunks are dropped.

LLMLingua-2. LLMLingua-2 (Pan et al., 2024) is a hard prompt compression method that deletes tokens from the context rather than replacing it with latent vectors. We use the released XLM-RoBERTa-large token classifier. Because the classifier never sees the question, each collection is compressed once and reused for every query, similar in principle to MEMORILLA. We concatenate the documents of the Full Context input with blank-line separators and compress them as one block, retaining newlines and question marks. The question, answer-format instruction, system prompt and chat template are copied verbatim, so the prompt differs from Full Context only in the compressed documents. We disable the option to discard whole documents, since document selection is retrieval and is already covered by RAG-k. We set absolute budgets of 32, 64, 128 and 350 tokens rather than

Table 5: **Downstream training and evaluation details.** Stage 3 trains on a unified downstream mixture. PV4 and PMv2 receive additional benchmark-specific fine-tuning, while all other benchmarks use the multi-task checkpoint directly.

Item	Configuration
Stage 1 corpus	1.98M EnWiki paraphrase reconstruction examples, chunked at approximately 128 tokens with mean 1.73 documents per row
Stage 2 corpus	Mixed supervision over SQuAD v2, DROP, CoQA, PubMedQA, QuAIL, PwC, CNN/DailyMail, SAMSum, DialogSum, and MS MARCO
Stage 3 corpus	Unified concatenation of PV4, PMv2, FactKG, TriviaQA, NarrativeQA, PubMedQA, LaMP-4, and LaMP-7
PV4 and PMv2 specialization	After multi-task fine-tuning, PV4 and PMv2 are each further fine-tuned for 5 epochs on the corresponding benchmark training split using the same recipe
Other downstream tasks	FactKG, TriviaQA, NarrativeQA, PubMedQA, LaMP-4, and LaMP-7 use the multi-task checkpoint directly with no additional task-specific training
Evaluation split	Official test split for every task except LaMP-4 and LaMP-7, which use their development splits; TriviaQA, which uses a deterministic seed-23 validation subsample with n=500; and PubMedQA, which uses the 100-example test portion of a 900/100 split derived from PQA-L

a compression rate, which the method treats only as a soft target, and report the best budget for each metric.

xRAG. xRAG (Cheng et al., 2024) maps the embedding of the top retrieved chunk to a single soft token in the input space of the decoder through a trained bridge. We reimplement the bridge in our codebase as a layer normalization followed by a two-layer MLP with a GELU activation, which maps the 2,560-dimensional chunk embedding to the 4,096-dimensional decoder input space with 27,276,288 trainable parameters. The chunk is retrieved by cosine similarity with the shared encoder. Only the bridge is trained, from scratch on each benchmark's training split with the same answer-token loss as MEMORILLA. We train for 5 epochs with AdamW at a learning rate of 10^{-4}, a weight decay of 0.01, a warmup ratio of 0.1, an effective batch size of 16, a maximum sequence length of 8,192 tokens and seed 23.

ComMer. ComMer (Zeldes et al., 2025) compresses user-history documents into latent representations with a learned compressor, merges them and passes them to a frozen LLM for personalized generation. We compare against its reported LaMP-4 and LaMP-7 results in Table 12.

Mem0 and MemoRAG. Mem0 (Chhikara et al., 2025) extracts and consolidates memories at write time, while MemoRAG (Qian et al., 2025) forms clue queries from a global memory to guide retrieval. We run the official implementations with the settings in Table 6. Both use a frozen Qwen3-4B to write their memory, as the fact extractor of Mem0 and as the memory model of MemoRAG. This matches the 4B scale of the Qwen3-Embedding-4B encoder with which MEMORILLA reads the same collections, so every method processes the documents with a 4B model and answers with the same frozen Qwen3-8B decoder. MemoRAG runs in its supported long-context memory mode, in which the memory model reads the uncompressed collection rather than a compressed version of it. Its retrieval uses each benchmark's own chunks and the shared encoder, so the clue mechanism is its only difference from RAG-k. We sweep the retrieval depth of Mem0 over $k \in \{10, 30, 50\}$ and of MemoRAG over its native setting and caps of 5 and 50 chunks, and report each at its best depth per metric.

D ADDITIONAL DETAILS ON BENCHMARKS

Personalization. These benchmarks require combining information spread across a user's past chats or writing into a picture of the user that no single document provides.

- **PersonaMem-v2** (PMv2, n=5,000) (Jiang et al., 2025). Each user has a long history of conversations with an assistant about everyday tasks, and each question asks for a response that fits the user's preferences. Most preferences are never stated outright but surface indirectly, for instance in an email the user asks to have polished, so the model must infer them from scattered cues across many sessions rather than look them up.
- **PersonalizationV4** (PV4, n=4,237). Our synthetic benchmark gives each user 200 two-turn conversations that reflect a persona without stating its traits (Appendix D.1). Each question places the user in a new scenario and requires combining at least two persona facts, which may appear in

Table 6: **Mem0 and MemoRAG configuration.** Rows below the rule are shared with the RAG-k baselines, so both methods differ from RAG only in the context given to the decoder.

Setting	Mem0	MemoRAG
Implementation	`mem0ai` 0.1.118 with its stock prompts and ADD/UPDATE/DELETE/NONE loop, unmodified	Official repository with its prompts unchanged
Memory-writing model	Frozen Qwen3-4B extracts facts and consolidates memories	Frozen Qwen3-4B reads the collection as its memory
Memory-model decoding	Greedy, up to 2,048 tokens	Greedy, up to 128 tokens per clue call
Write path	One `add()` per document, with each extracted fact reconciled against its 5 nearest stored memories	Collection read once per entity and cached, front-truncated only where it exceeds the context window (NarrativeQA)
Memory scope	One Qdrant collection per user (PMv2, PV4), story (NarrativeQA) or question (others)	One memory per user, story or question
Retrieval query	The question	Clue spans and surrogate questions longer than 3 words, plus the question
Retrieval	Top-k memories by cosine similarity	Top 3 chunks per query, unioned, deduplicated and kept in corpus order
Depths swept	$k \in \{10, 30, 50\}$ memories	Native (about 10 to 20 chunks), and capped at 5 and 50 by maximum similarity over queries
Memory-model answer	Not applicable	Not used (`use_memory_answer=False`)
Retriever	Qwen3-Embedding-4B with 2,560 dimensions and no instruction prefix, over each benchmark's chunks (Table 2)	
Generator	Frozen Qwen3-8B with the system prompt, chat template, answer-format instruction and maximum new tokens of RAG-k	
Decoding	Greedy, with T=0, top-p=1.0 and top-k=-1	

different conversations. A hard subset (n=629) keeps the questions that the untrained Qwen3-4B decoder answers incorrectly without documents or with the top retrieved conversation.

- **LaMP-4** (n=1,925), **LaMP-7** (n=1,500) (Salemi et al., 2023). LaMP-4 asks the model to write a headline for a new news article in the style of the author's earlier articles and headlines, and LaMP-7 asks it to rewrite a tweet in the style of the user's earlier tweets. The challenge is stylistic rather than factual, since style emerges from the whole writing history rather than a single item.

Long-Document QA. This axis tests whether memory can compress entire long documents into a usable representation for downstream QA.

- **NarrativeQA** (NQA, n=10,557) (Kočiský et al., 2017). Questions concern full books and movie scripts and were written from plot summaries without access to the stories, so they target events, motives and relations described across the narrative rather than sentences that can be matched locally. Answers are free-form text, which rewards understanding the plot over copying a passage.
- **TriviaQA** (n=500 from the 17,944-question validation split) (Joshi et al., 2017). Questions written by trivia enthusiasts are paired with Wikipedia and web documents collected afterwards. Since the questions were not written against these documents, their wording and structure often differ from the passages that answer them, and an answer may require combining facts across sentences.

Knowledge-Intensive QA. These benchmarks test factual grounding from structured or semi-structured evidence of varying quantity and format.

- **FactKG** (n=9,041) (Kim et al., 2023). Each claim must be labeled as supported or refuted by facts from the DBpedia knowledge graph, through one-hop lookups, conjunctions of several facts, existence checks, multi-hop chains or negations. A multi-hop claim such as "AIDAstella was built by a company in Papenburg" requires chaining two facts, neither of which verifies the claim on its own, and a negated claim requires confirming that a relation is absent.
- **PubMedQA** (n=100) (Jin et al., 2019). Each question is a biomedical research question based on an article title, answered yes, no or maybe from the article's abstract with its conclusion removed. The model must infer the conclusion from the reported methods and results, often by interpreting quantitative findings. The entire source fits easily in the context, which makes PubMedQA a contrast case where condensing the source offers little benefit.

Table 7 summarizes the benchmark datasets evaluated in this work. All datasets use standard train/validation/test splits. For the personalization benchmarks (PV4, PMv2, LaMP-4, LaMP-7), training and evaluation users are strictly disjoint to prevent leakage of user-specific information. We

evaluate LaMP-4 and LaMP-7 on their development splits and TriviaQA on the validation split. For PubMedQA we use a 900/100 train/test split derived from the PQA-L source. Table 10 reports how many document tokens each method places in the decoder per query. The serving stack enforces a 32,768-token decoder limit and enables YaRN for every full-context example that exceeds it, which covers part of PMv2, TriviaQA and NarrativeQA and all of PV4. Every PubMedQA collection and 96.7% of LaMP-7 collections hold at most 50 chunks, so RAG-50 reads the same chunks as Full Context on them.

Table 7: **Benchmark datasets overview.** Each row corresponds to an evaluation benchmark. Each data point consists of a textual query paired with a collection of source documents from which the model must produce an answer or generation. $\bar{\mu}$ docs/q and $\bar{\mu}$ tok/q report the mean number of source documents and total source tokens per query in the test split.

Dataset	Task	$\bar{\mu}$ docs/q	$\bar{\mu}$ tok/q	Train	Val	Test
PersonalizationV4 (PV4) (Ours)	Personalization SA	200	50,476	15,184	1,769	4,237
PersonaMem-v2 (PMv2) (Jiang et al., 2025)	Personalization SA	116	31,618	18,521	2,059	5,000
NarrativeQA (NQA) (Kočiský et al., 2017)	Long-document generative QA	1,420	76,225	32,747	3,461	10,557
FactKG (Kim et al., 2023)	KG fact verification	129	1,665	86,367	13,266	9,041
TriviaQA (Joshi et al., 2017)	Open-domain factoid QA	164	11,861	138,384	17,944	—
PubMedQA (Jin et al., 2019)	Biomedical QA	4	324	900	—	100
LaMP-4 (Salemi et al., 2023)	Personalized headline gen.	172	7,638	12,527	—	1,925
LaMP-7 (Salemi et al., 2023)	Personalized tweet paraphrase	18	465	10,437	—	1,500

D.1 PERSONALIZATIONV4 DATA PIPELINE

PersonalizationV4 is a synthetic benchmark for evaluating multi-document user modeling. Each user has one LLM-generated persona profile seeded from the Faithful Persona-based Conversational Dataset (Jandaghi et al., 2023). The resulting profiles are 10k to 22k characters long (median 15.5k) and describe demographics, personality, work, hobbies, communication style and how preferences change over time. All of the user's data is derived from this profile through the five steps below, and Table 8 traces one held-out user through each of them.

1. **Question categories.** GPT-5.2 reads the persona and extracts 10 persona-specific categories that questions can test.
2. **Chat topics.** GPT-5.2 writes exactly 200 one-sentence scenarios per user in which the user asks the assistant for help, covering every part of the persona.
3. **Dialogues.** GPT-5-mini expands each topic into one two-turn dialogue with one user turn and one assistant turn. The user embodies the persona implicitly without stating its traits. Exactly 200 dialogues are generated per user and all form the collection that the memory module reads; there is no truncation step.
4. **Questions.** For each category, GPT-5.2 writes 13 to 15 questions from the persona alone, with 14 questions in 1,292 of the 1,560 category files. Each question has a reference answer and four distractors. The instructions in Table 9 require every question to combine at least two persona facts in a new scenario and every distractor to be a plausible preference that matches the reference answer in length, structure and specificity. Outputs that cannot be parsed are discarded, leaving 133 to 145 questions per user (mean 141.3).
5. **Randomization.** The reference answer's position among its distractors is shuffled with a fixed seed of 42.

Questions are posed without their candidate answers, and the model answers in free text. The reference answer and distractors are used only by the short-answer scorer, which counts a response as correct when its nearest candidate under the Qwen3-Embedding-4B encoder is the reference answer. We release 156 users, train on 120 users and evaluate on 30 held-out users, none of whom appear in training. Every question for these held-out users is evaluated: 139 to 143 per user and 4,237 in total, comprising 3,793 questions from the users' train split and 444 from their test split. The hard subset (n=629) is defined from the untrained Qwen3-4B decoder's multiple-choice answers and contains questions it answers incorrectly without documents, with the top-1 retrieved document, or in both settings; 299 of the 629 questions are answered incorrectly in both settings.

Table 8: **One held-out PersonalizationV4 user through the pipeline.** Excerpts are verbatim apart from abridgement. The question draws on base-building, which appears only in the dialogue shown, and on the Iron Heart project, which appears in 18 other dialogues, so answering requires combining documents. The distractors are never shown to the model and serve only as scoring candidates.

Persona	Name Maya Patel. Age 34. Location Austin, Texas, USA. Occupation Certified Welder & Part-Time CNC Operator at a regional aerospace supplier. … Maya grew up in a multicultural household where her Indian-American parents emphasized discipline, education, and the importance of physical well-being. After completing a two-year technical diploma in welding technology, she entered the workforce in a traditionally male-dominated industry, where she quickly earned a reputation for precision, reliability, and an unflappable calm under pressure. Though she never pursued a university degree, her practical knowledge of metallurgy, robotics-aided fabrication, and safety protocols rivals that of many engineers. …
Categories	Aerospace welding expertise & mental rehearsal habits. Gaming genre preferences & playstyle strategies. Morning mango ritual & mobility routine. … (3 of 10)
Chat topic	Maya asks for a list of cooperative building-and-craft games she can play online with classmates after welding class.
User turn	… looking for coop building-and-craft games I can jump into with the welding class after tonight's session. Need stuff that's accessible for mixed skill levels, supports voice chat, and has meaningful base-building or automation mechanics. Toss me a short list with platform notes and why each one fits.
Assistant turn	*(seven games with platform notes, from Minecraft to Raft)* … Each supports voice via Discord/Steam, scales with mixed experience, and offers clear building goals for group sessions.
Question	In a new open-world crafting RPG, players can specialize in one mastery for unique bonuses. Considering Maya's welding expertise, love for base-building, and ergonomic focus from the Iron Heart project, which specialization best aligns with her likely in-game identity?
Reference answer	Master Builder, optimizing structural stability, workstation placement, and production comfort, because she prefers engineering-style layouts that echo her fabrication and ergonomic design background.
Distractors	Arcane Historian, optimizing lore discovery and dialogue outcomes, because she prefers uncovering story fragments over experimenting with materials and structural layouts. Beast Whisperer, optimizing pet behavior and breeding trees, because she prefers managing biological companions over designing durable and efficient physical constructs. Shadow Scavenger, optimizing stealth routes and loot efficiency, because she prefers sneaking past threats over methodically constructing and upgrading comprehensive bases. Combat Tactician, optimizing battlefield formations and ambushes, because she prefers front-line tactics over refining structures, workstations, or long-term infrastructure planning.

Table 9: **Question-generation instructions.** Verbatim excerpts from the prompt given to GPT-5.2 for each category, which also contains the full persona profile, the target category and the number of questions to write.

Questions	Each question must require synthesizing information or understanding nuance from the persona. Questions must require combining two or more facts from the persona profile. Avoid questions that can be answered by recalling a single fact. Embed each question in a specific, realistic scenario that requires applying persona knowledge to a novel situation rather than simple recall. The question stem must not hint at characteristics of the answer.
Distractors	Every choice (A-E) must be a reasonable, professional, or logical preference for a human. Never use obviously wrong, absurd, or incompetent distractors. Ensure distractors include industry-standard best practices that most people in the persona's role would choose, but which this specific persona rejects. Use information that is true about the persona but applies it to the wrong situation or frequency. All five choices must have the same word count within a tolerance of two words.

D.2 Context Budget per Query

Table 10 reports the number of document-derived tokens each method places in the decoder for one query. We exclude the question, the answer-format instruction, the system prompt and the chat template, which are identical across methods. For each text baseline, the count is the length of its prompt minus the length of the closed-book prompt for the same question, measured with the Qwen3-8B tokenizer and averaged over the evaluated questions. Closed-book receives no document tokens, MEMORILLA contributes its 16 memory tokens and xRAG a single token. Figure 1 places each method at the geometric mean of these counts over the eight benchmarks, and the compression column of Table 11 divides the Full Context count by the 16 tokens of MEMORILLA.

Table 10: **Document tokens per query.** Mean number of document-derived tokens in the decoder input, excluding the question and all shared instructions. Mem0, MemoRAG and LLMLingua-2 are counted at the setting reported for each benchmark's primary metric in Table 11, and truncated inputs are counted as read. Fits 32K reports whether every Full Context prompt fits the enforced 32,768-token limit, and Partial means that YaRN is enabled for the remaining examples. † marks benchmarks where RAG-50 reads the same chunks as Full Context, for every PubMedQA question and 96.7% of LaMP-7 questions.

Benchmark	Docs/q	Fits 32K	RAG 1	RAG 5	RAG 50	Full Context	LLM Lingua-2	xRAG	Mem0	Memo RAG	MEMORILLA
PMv2	116	Partial	125	652	9,009	31,587	250	1	665	9,254	16
PV4	200	No	201	1,021	10,832	50,455	180	1	978	11,046	16
PubMedQA	4	Yes	69	321	324†	324	290	1	8	324	16
NarrativeQA	1,420	Partial	69	354	3,612	67,385	596	1	493	800	16
TriviaQA	164	Partial	51	226	2,357	11,888	243	1	269	303	16
FactKG	129	Yes	13	64	568	1,727	279	1	300	573	16
LaMP-4	172	Yes	45	227	1,715	7,580	227	1	251	230	16
LaMP-7	18	Yes	25	125	435†	459	299	1	81	135	16
Geometric mean			55	273	1,808	6,301	278	1	212	784	16

E ADDITIONAL RESULTS AND ABLATIONS

Table 11 reports the complete 8B results underlying the main quantitative comparison in Figure 1. Table 12 compares MEMORILLA with ComMer on LaMP-4 and LaMP-7.

Table 11: **Main results across eight benchmarks.** All methods use a frozen Qwen3-Embedding-4B encoder and a frozen Qwen3-8B decoder; Mem0 and MemoRAG also use a frozen Qwen3-4B to build their memories. RAG-k retrieves the top-k chunks by cosine similarity and Full Context reads the full collection. MEMORILLA uses $K{=}16$ memory tokens. Best result per row is **bold**, second best is underlined. The Compression column reports how many times more document tokens Full Context gives the decoder than the 16 memory tokens of MEMORILLA (Table 10). LLMLingua-2 was not evaluated on the PV4 hard subset.

Task	Metric	Closed book	RAG 1	RAG 5	RAG 50	Full Context	LLM Lingua-2	xRAG	Mem0	Memo RAG	MEMORILLA	Compression
PMv2	Acc (SA)	25.64	26.10	25.92	26.72	27.92	26.92	30.90	27.48	28.76	**44.60**	1,974×
PV4	Acc (SA)	58.30	62.21	64.98	65.78	59.12	52.14	62.64	65.07	**67.88**	66.65	3,153×
	Acc (SA, Hard)	31.96	38.00	39.27	41.97	36.57	—	34.98	44.20	**45.95**	41.97	
PubMedQA	Acc	21.00	34.00	67.00	67.00	67.00	**69.00**	36.00	24.00	67.00	59.00	20×
NarrativeQA	BLEU-1	7.09	7.36	7.85	8.29	5.89	6.87	**16.86**	7.77	9.14	16.36	4,212×
	ROUGE-1	11.54	13.32	15.60	18.45	19.12	7.26	17.33	14.24	**21.80**	19.68	
	ROUGE-L	10.71	12.44	14.59	17.23	17.99	7.18	16.92	13.80	**20.51**	19.31	
	F1-Score	9.87	10.51	11.64	13.27	13.50	5.76	15.22	11.58	15.03	**16.56**	
TriviaQA	EM	36.00	51.80	55.20	55.60	52.60	42.40	47.00	44.60	56.60	**57.00**	743×
	BLEU-1	29.12	40.29	39.09	43.07	35.56	29.28	46.54	29.09	45.00	**58.66**	
	BLEU-4	3.74	8.45	7.54	10.94	9.15	4.30	11.31	5.54	10.35	**14.79**	
	F1-Score	37.68	52.06	54.99	58.05	56.49	39.74	50.37	42.74	57.55	**60.60**	
FactKG	Acc	65.97	73.93	76.95	78.39	80.43	71.65	78.98	72.79	80.38	**84.33**	108×
LaMP-4	ROUGE-1	14.24	16.15	16.98	16.98	16.21	14.98	18.12	14.72	17.08	**20.30**	474×
	ROUGE-L	12.24	14.31	14.94	15.09	14.21	12.98	16.10	12.70	15.02	**18.15**	
	F1-Score	8.98	11.21	12.15	12.15	11.32	8.71	13.46	9.05	12.41	**15.54**	
	METEOR	7.13	9.78	10.95	10.95	9.85	6.99	10.11	7.28	11.21	**11.41**	
LaMP-7	ROUGE-1	48.73	49.80	49.80	49.65	49.57	49.61	52.78	49.58	49.92	**54.40**	29×
	ROUGE-L	43.13	44.11	43.81	43.77	43.71	43.73	46.84	43.68	44.00	**48.37**	
	F1-Score	36.10	37.28	37.28	37.26	37.14	37.17	42.99	36.87	37.55	**44.96**	
	METEOR	32.31	33.70	33.78	33.73	33.55	33.74	38.35	33.30	34.12	**40.37**	

E.1 MEMORY ARCHITECTURE

We port the BLIP-2 bridging module (Li et al., 2023) to document collections and report it as a **Q-Former** architecture ablation. Table 13 compares methods for constructing 16 memory slots from

N document embeddings on PMv2. Each method is trained from scratch using the hyperparameters in Table 4.

Global mean-pool. This baseline averages all document embeddings into one collection vector and copies it into all 16 slots.

Segmented mean-pool. This baseline partitions the ordered sequence of document embeddings into 16 adjacent groups of documents and averages each group to form one slot. The groups contain complete documents rather than subdivisions of individual documents.

Segmented with matched parameters. This method begins with the 16 slots produced by segmented mean-pooling and increases capacity to match MEMORILLA without learned queries or query conditioning.

Cosine top-16 with matched parameters. This method selects the 16 document embeddings with the highest cosine similarity to the question and uses them as the memory slots. Its slot processor matches the parameter budget of MEMORILLA.

Documents only. This Q-Former variant self-attends over its learned latent slots and then cross-attends to all document embeddings without access to the question.

Q-Former. This method self-attends over the learned latent slots and question embedding before the slots cross-attend to all document embeddings.

MEMORILLA. Our method first conditions the learned slots on the question embedding and then uses the conditioned slots to cross-attend to all document embeddings. It does not use self-attention.

Query-independent pooling is weakest and parameters do not fix it, since segmented pooling at matched budget reaches 31.10 on short-answer against 44.04. The Q-Former, which also conditions on the question, nearly matches MEMORILLA on PMv2 (43.34 against 44.04). On FactKG, MEMORILLA also outperforms the Q-Former (82.56 against 80.48), despite the Q-Former using 373,762,560 parameters. The documents-only variant reaches 78.72 on FactKG with 373,762,560 parameters. Replacing every document embedding with Gaussian noise changes the documents-only result by only +0.04 while lowering MEMORILLA by 5.93 and the Q-Former by 5.11. The gain therefore comes from conditioning the slots before they read rather than from either interaction on its own.

E.2 MEMORY CAPACITY AT TRAINING TIME

Figure 9 compares models trained from scratch at $K{=}16$ and $K{=}128$ under otherwise identical settings. All runs use the Qwen3-4B decoder and the same hyperparameters reported in Table 4, with K as the only changed hyperparameter. Between $K{=}16$ and $K{=}128$, four of eight benchmarks move by less than a point at eight times the capacity and PMv2 and PV4 drop, so we set $K{=}16$ throughout. All runs are trained from scratch, so this concerns capacity rather than the deployed checkpoint.

E.3 MEMORY TOKEN BUDGETS FOR INFERENCE

Figure 10 varies the number of memory tokens K at inference time without retraining. Both personalization benchmarks gain most from the last slots. PMv2 stays near its closed-book level through $K{=}4$ (26.58 against 25.64) and climbs to 44.60 at $K{=}16$, while PV4 starts near its closed-book level (57.66 against 58.30), dips at $K{=}2$ and 4, and climbs to 66.65. Neither shows saturation, which reflects slots dropped from a checkpoint trained at $K{=}16$ rather than headroom above it, since training at $K{=}128$ does not help. FactKG, TriviaQA and NarrativeQA rise with every doubling.

Table 12: **Comparison with ComMer** on personalized generation, against ComMer's reported results. ComMer uses Gemma-2b while MEMORILLA uses Qwen3.

Method	**Tok.**	**LaMP-4** ROUGE-L	**LaMP-7** ROUGE-L
ComMer	128	17.50	45.00
MEMORILLA (4B)	16	<u>17.72</u>	<u>47.04</u>
MEMORILLA (8B)	16	**18.15**	**48.37**

Table 13: **Memory architecture ablation.** All methods use the frozen Qwen3-Embedding-4B encoder and Qwen3-4B decoder and share the training recipe, prompts, decoding and scorer. They are trained from scratch, so values are not comparable to Table 11. The Q-Former variants use more parameters than MEMORILLA.

Method	Query aware	Params	PMv2 SA
Global mean-pool	no	6,566,400	28.94
Segmented mean-pool	no	6,566,400	29.68
Segmented, parameter-matched	no	190,149,120	31.10
Cosine top-16, parameter-matched	yes (hard)	190,149,120	33.86
Documents only	no	268,856,320	34.72
Q-Former	yes (soft)	268,856,320	43.34
MEMORILLA	yes (soft)	190,190,080	**44.04**

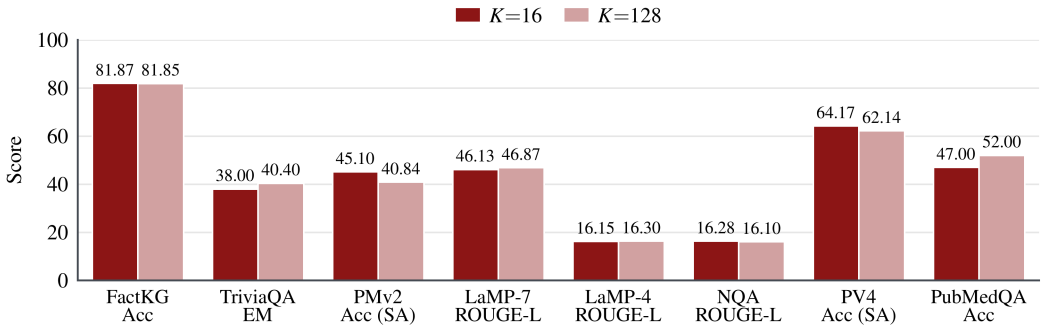


Figure 9: **Memory-token capacity.** All models use a Qwen3-4B decoder and are trained from scratch under identical settings, differing only in the number of memory tokens K (K=16 is the setting used throughout).

PubMedQA and LaMP-4 remain roughly flat at low K and improve only from K=8, and LaMP-7 only from K=16. Across all three regimes, the required memory capacity tracks the degree of cross-document integration each task demands.

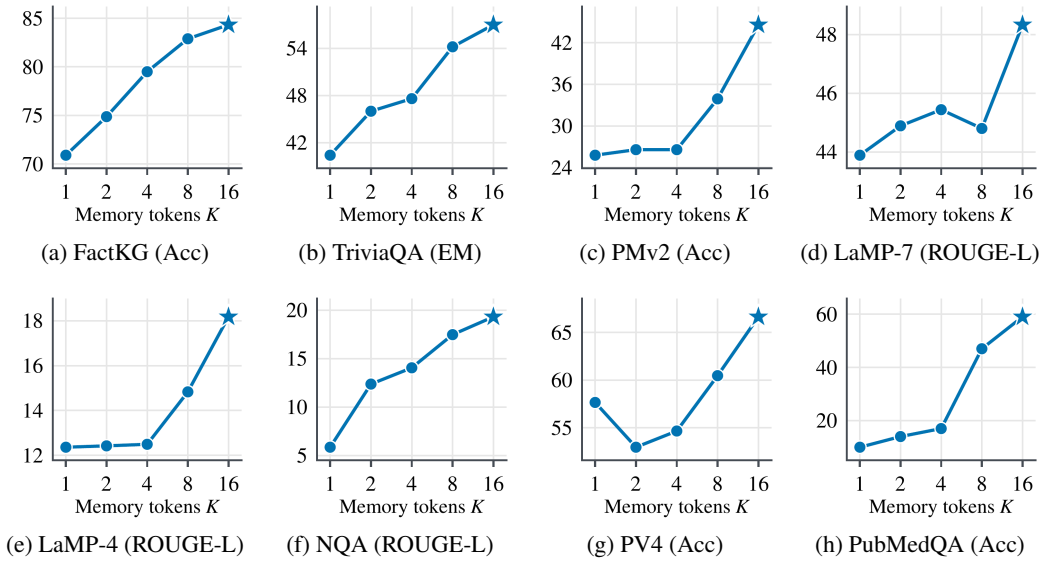


Figure 10: **Memory token budget at inference.** Performance of MEMORILLA with a frozen Qwen3-8B decoder using the first K memory tokens of the checkpoints trained with the full 16 tokens budget.

E.4 Frozen vs. LoRA Training

To separate the memory interface from generic adaptation, we train rank-130 LoRA adapters (Hu et al., 2021) on the Qwen3-4B decoder with 268.4M parameters, comparable to the 194.1M parameters of the memory module paired with the 8B decoder. The adapters use the same training rows as MEMORILLA's per-task fine-tuning and receive no documents in context. Figure 11 shows that MEMORILLA leads on every primary metric, although the PubMedQA and LaMP-7 gaps are within noise. Together with the noise test above, where MEMORILLA loses 5.93 on FactKG without real documents, this indicates that its gains rely on the stored evidence rather than on adaptation alone. Freezing both backbones keeps the store portable and the decoder untouched, at the cost of an encoder never optimized for compression.

E.5 Role of Encoder Size

Figure 12 swaps the frozen document encoder from Qwen3-Embedding-4B to Qwen3-Embedding-8B, keeping the Qwen3-8B decoder, memory module and recipe fixed and training from scratch. Every score moves by at most 1.3 except PubMedQA, where the +3.00 gain corresponds to three of 100 questions, so a larger encoder from the same family has minimal effects on performance.

F Computational Analysis of Memorilla

Consider a collection of N documents serving Q queries. MEMORILLA pays the encoder once on the write path, producing N embeddings that persist and are reused by every later query. Dense retrieval builds the same index, so this cost is shared rather than added. The two methods differ on the read path. RAG places the selected text in the decoder, so its prefill grows with retrieval depth, while MEMORILLA runs one cross-attention read over the stored embeddings and always hands the decoder $K{=}16$ tokens. Table 14 reports measurements on one H100-80GB at batch size 1 with a Qwen3-8B decoder. Growing the collection by 512$\times$ decreases the memory-module forward time by 0.06 ms and MEMORILLA end-to-end latency by 1.9 ms, while full-context decoding grows by 19,184 ms and exceeds the enforced 32,768-token limit, requiring YaRN. Transfer stays under 0.04% of query latency because a 200 document collection is 1 MB, and the store is 7,266$\times$ smaller than the corresponding KV cache on PV4 (50,455 document tokens at 144 KiB per token for Qwen3-8B in bf16, 6.9 GiB, against 200 embeddings of 2,560 bf16 values). Absolute latency is dominated by decoding rather than by memory, so the benefit is not a faster single query but a serving cost that stays flat as collections grow and amortizes across the Q queries that read them.

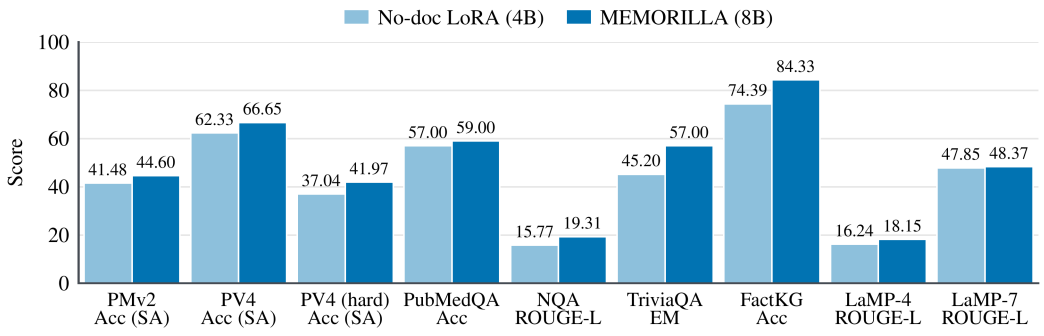


Figure 11: **Memory versus comparable-budget adaptation.** MEMORILLA with an 8B decoder and 194.1M trainable parameters against a no-document rank-130 LoRA adapter with a 4B decoder and 268.4M trainable parameters.

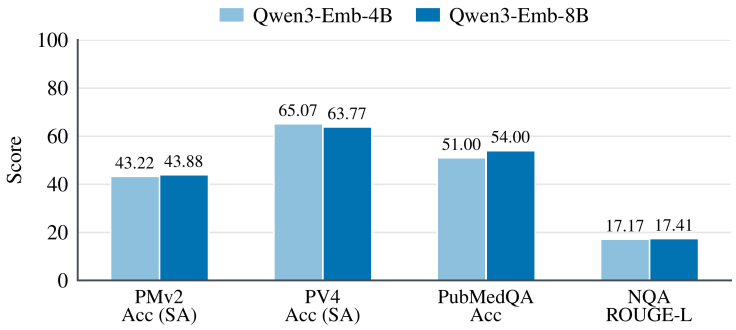


Figure 12: **Encoder size.** The memory module is trained from scratch with a frozen Qwen3-8B decoder and either encoder; all other settings are fixed.

Table 14: **Serving cost as the collection grows.** One NVIDIA H100-80GB, batch size 1, bf16, greedy, `max_new_tokens`=32, 250-token documents; median of 5 timings after 2 warmups. The write path is paid once per document and is shared with dense retrieval; read path and decoder cost are paid per query. Full Context at N=512 is 128,064 context tokens, past the enforced 32,768-token limit, so it uses YaRN (factor 4). Timings use HF transformers `generate` without a paged KV cache, so Full Context latency depends on the serving stack.

Path	Quantity	N=1	N=512	Growth
Write	Document encoding	22.92 ms	2,588.84 ms	linear in N
	Storage per document	5 KB	5 KB	flat
Read	Memory module forward	2.01 ms	1.96 ms	−0.06 ms
	CPU to GPU transfer	0.014 ms	0.060 ms	+0.046 ms
Decoder	Context tokens	16	16	flat
	MEMORILLA end to end	740 ms	739 ms	−1.9 ms
	Full Context end to end	749 ms	19,932 ms	+19,184 ms

G CONTINUAL LEARNING WITH MEMORILLA: ADDITIONAL DETAILS

TextWorld games require an agent to maintain state across a partially observable trajectory, navigate a map, manipulate objects, and execute ordered action sequences. Observations are room descriptions and action outcomes, actions are short text commands, and games span up to 65 rooms. From the second turn on, MEMORILLA embeds each completed turn as one document, holding its observation truncated to 200 characters and its action (74 tokens on average), reads these documents with the current observation as the query, and places the resulting K=8 memory vectors before the prompt. The 20 extreme, huge, long and mega-ultra test games instead embed one document per five completed turns with full observations, keep the last four documents, and use the mean document embedding as the query. Figure 13 shows a prompt and the documents compressed into the memory vectors, and Section 6.1 reports how the trajectory grows across a rollout.

We train on 800 prompts and evaluate on held-out validation and test splits of 40 games each (30 distinct, since the five medium and five hard games appear twice). Games come in six tiers (medium, hard, long, mega-ultra, huge, extreme) with 15–65 rooms and 6–74-step quests; the 800 training prompts are 202 medium, 558 hard and 10 from each longer tier. Rollouts sample up to 50 environment turns with at most 128 new tokens per turn, and evaluation samples three completions per held-out game at temperature 0.6 using the same SkyRL (Cao et al., 2025) generation harness for every condition. The random-memory baseline uses the MEMORILLA architecture with untrained weights, so it adds eight latents to the same prompt without having learned what to place in them. Table 15 lists the full configuration.

Decoder input when choosing the turn-6 action ([P] marks the eight memory positions)

```
[P][P][P][P][P][P][P][P]<|im_start|>system
You play TextWorld efficiently.
Use memory summaries when useful and avoid repeating failed actions.
Return one final action command in the format [ACTION: <command>].<|im_end|>
<|im_start|>user
What is your next action?<|im_end|>
<|im_start|>user
<ASCII title banner, objective and first room description><|im_end|>
<|im_start|>assistant
... [ACTION: go south]<|im_end|>
<|im_start|>user
-= Laundromat =- You've entered a laundromat. ...<|im_end|>
   ... turns 2-5: assistant actions and user observations ...
<|im_start|>user
-= Canteen =- You arrive in a canteen. ...<|im_end|>
<|im_start|>assistant
```

Memory documents embedded at this turn (observations cut to 200 characters)

```
Turns 1-1 | Reward: 0.0 | Score Change: 0
Turn 1:
Obs: <first 200 characters of the ASCII title banner>
Act: go south

Turns 2-2 | Reward: 0.0 | Score Change: 0
Turn 2:
Obs: -= Laundromat =- You've entered a laundromat. You make out a case. ...
     You don't like doors? Why not try going east, that entranceway is
     unblocked. You d
Act: go south

   ... documents 3 (Bedchamber, go south) and 4 (Salon, go south) ...

Turns 5-5 | Reward: 0.0 | Score Change: 0
Turn 5:
Obs: -= Study =- You arrive in a study. ... There is an unguarded exit to
     the east. There is an unblocked exit to the north.
Act: go east
```

Query and action

Query: the full turn-6 observation ("-= Canteen =- You arrive in a canteen. A typical one. ..."). **Output:** `[ACTION: go east]`

Figure 13: **An example TextWorld prompt.** A winning 12-turn rollout on a held-out medium game, at the choice of the turn-6 action; training prompts have the same format. The eight memory vectors sit before the system prompt and are recomputed every turn from one document per completed turn, with the current observation as the query, while the full trajectory stays in the prompt. Line breaks inside the documents are added for width.

G.1 Reinforcement-Learned Memory Objective

For agentic settings, we train the memory module from environment rewards while keeping the encoder and decoder frozen (Figure 7). At each TextWorld turn, the interaction trajectory is embedded as a document collection, f_{mem} maps it to memory vectors, and the frozen decoder samples the next action. We optimize only $\boldsymbol{\theta}_{\text{mem}}$ with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a critic-free variant of PPO (Schulman et al., 2017). For each prompt, a group of G rollouts receives scalar rewards R_i from the environment, and advantages are standardized within the group:

$$A_i = \frac{R_i - \text{mean}_{j \in G}(R_j)}{\text{std}_{j \in G}(R_j) + \epsilon}. \tag{8}$$

The memory policy is then updated with the clipped PPO surrogate over generated action tokens,

$$\mathcal{L}_{\text{RL}}(\boldsymbol{\theta}_{\text{mem}}) = -\mathbb{E}\left[\min\left(r_i A_i,\ \text{clip}(r_i, 1-\epsilon_c, 1+\epsilon_c) A_i\right)\right], \quad r_i = \frac{\pi_{\boldsymbol{\theta}_{\text{mem}}}(a_i \mid \tau_i)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_i \mid \tau_i)}. \tag{9}$$

The reward is the TextWorld game score, without shaping: medium and hard games score only on winning, while long and mega-ultra games also score as each of their quests is completed. The

injection mechanism is unchanged from the supervised setting, while the memory module adds one self-attention layer over the trajectory documents before its cross-attention blocks. Memory here supplements the trajectory rather than replacing it, distinguishing it from the compression results in Section 4. At turn t, the decoder input follows Eq. (6) with the query replaced by the trajectory $\tau^{(t)}$, which contains the game description and all prior observations and actions:

$$\mathbf{X}_{\text{LLM}}^{(t)} = [\mathbf{m}_1^{(t)}, \ldots, \mathbf{m}_K^{(t)}; \mathbf{P}_{\tau^{(t)}}], \tag{10}$$

where the memory tokens $\mathbf{m}_k^{(t)}$ are recomputed at every turn after the first from the completed turns of $\tau^{(t)}$, with the current game observation as the query q (the mean document embedding in the longer games).

Table 15: **TextWorld RL configuration.** The memory module is the only trainable component, and the full textual history is retained in every condition for a fair comparison.

Component	Setting	Value
Environment	Task suite	TextWorld held-out games up to 65 rooms
	Splits	800 train prompts; 40-game validation and test splits
	Agent state	Initial description plus prior observations/actions
	Reward	TextWorld game score; no shaping reward
Model interface	Decoder	Frozen Qwen3-4B
	Trajectory encoder	Frozen Qwen3-Embedding-4B
	Memory budget	K=8 latent vectors inserted into decoder prompt
	Memory architecture	8 heads; 1 self-attention layer; 2 cross-attention layers
	Context parity	16,384-token window; full history kept for ICL
Optimization	Trainable parameters	Memory module only; decoder and encoder frozen
	Initialization	Random (Xavier-uniform slots); not loaded from the supervised checkpoint
	RL algorithm	GRPO; group size G=16; normalized group advantages
	Policy objective	PPO clip 0.2; token-mean loss; no critic/KL/entropy
	Optimizer	AdamW; learning rate 5×10^{-4}; no warmup
Rollouts	Training batch	4 prompts/step; 16 sampled rollouts/prompt
	Horizon	Up to 50 environment turns/rollout
	Action generation	Up to 128 new tokens/turn with SkyRL
Evaluation	Sampling	3 samples/held-out game at temperature 0.6
	Metrics	avg@3 mean solve rate; pass@3 any-sample solve rate