May the Memory Be With You: Efficient and Infinitely Updatable State for Large Language Models

Excel Chukwu
echukwu@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Saarland, Germany

Laurent Bindschaedler
bindsch@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Saarland, Germany

Abstract

Large language models (LLMs) excel at natural language tasks but lack persistent state management for personalized and adaptive interactions. We propose a framework that endows these models with stateful capabilities by combining retrieval-augmented generation (RAG) and low-rank adaptation (LoRA). Our approach recasts the LLM as an editable component that retains hierarchical knowledge, analogous to deferred merge operations in log-structured merge (LSM) trees. The system integrates short-term context with long-term memory by storing accumulated context in a retrieval system while periodically training and combining lightweight LoRA adapters. Preliminary evaluations demonstrate improved state retention and query performance compared to both standard LLMs and RAG-augmented models, supporting our vision for scalable, stateful AI systems.

CCS Concepts

• **Information systems** → *Information retrieval*; **Data management systems**; **Data structures**; • **Computing methodologies** → **Natural language processing**; *Machine learning*; **Knowledge representation and reasoning**; • **Software and its engineering** → *Software system structures.*

Keywords

Large language models, stateful AI systems, low-rank adaptation, retrieval-augmented generation, contextual knowledge management, persistent state retention, hierarchical state management, log-structured merge trees.

EuroMLSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1538-9/2025/03
https://doi.org/10.1145/3721146.3721951

ACM Reference Format:
Excel Chukwu and Laurent Bindschaedler. 2025. May the Memory Be With You: Efficient and Infinitely Updatable State for Large Language Models. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3721146.3721951

1 Introduction

Large language models (LLMs) are now deployed as the central component in various applications, ranging from conversational agents to complex decision support systems. These models have demonstrated exceptional capabilities in natural language understanding and generation; however, they inherently lack mechanisms for persistent state management. In many applications, retaining user-specific context, historical task data, and evolving situational information is crucial to enable personalized and adaptive interactions.

This paper addresses the core problem of effective retention, retrieval, and management of contextual knowledge (state) in AI systems that rely on LLMs. Existing approaches such as retrieval augmented generation (RAG) [8, 13], differentiable memory [5, 21], and in-model knowledge editing [6, 16, 23] have been proposed to overcome the limitations of statelessness, but they suffer from significant limitations. RAG methods excel at storing extensive information and retrieving it based on semantic similarity. However, they often struggle to retrieve contextually nuanced details when the query lacks strong semantic alignment with the stored representations, and they are limited by the size of the model's contextual window. On the other hand, differentiable memory architectures and knowledge editing techniques allow for direct updates to the model's internal knowledge but come with significant computational costs and scalability limitations and are not well-suited for handling dynamic, continuously evolving state information.

We present a novel hybrid framework that combines RAG with low-rank adaptation (LoRA) techniques [4, 10] to overcome the limitations of existing methods. The key intuition behind our approach is to reconceptualize the LLM as a stateful, editable component that retains knowledge in a hierarchical manner. By storing the accumulated state in

a retrieval system and periodically training corresponding lightweight LoRA adapters, our method maintains an efficient and continuously updated representation of the user context that supports rapid and precise retrieval even as the stored information grows, ensuring accurate and scalable retrieval with low computational overhead.

Our framework decomposes persistent state into modular components that are managed by specialized adapters. These adapters are designed to handle various aspects of the state, such as user preferences, task histories, and domain-specific instructions. A hierarchical management mechanism governs the consolidation and merging of overlapping or outdated information, enabling dynamic loading and multi-level state retrieval. This approach parallels the functioning of LSM trees, wherein recent data is maintained in fast memory and periodically merged into longer-term storage [14, 17].

Our preliminary experimental evaluation demonstrates that the proposed framework significantly improves state retention and query performance relative to traditional approaches. The initial prototype supports our vision by showcasing improved efficiency in computational cost and retrieval accuracy. These early results provide evidence that integrating RAG with LoRA adapters is a viable direction for building stateful AI systems in dynamic scenarios.

In summary, this paper makes the following contributions:

- We present a novel framework that transforms stateless LLMs into stateful systems by combining RAG and LoRA adapters.
- We propose a hierarchical state management mechanism that efficiently organizes and consolidates evolving contextual information, drawing analogies to LSM.
- We provide empirical evidence that our prototype improves state retention, computational efficiency, and response accuracy compared to traditional approaches.

In the remainder of this paper, we describe our system design and implementation, present preliminary results that support our vision, and discuss limitations and future work.

2 Background and Motivation

This section provides a brief overview of key concepts and related work that motivate our design choices. We discuss context retention in LLMs, retrieval-augmented generation (RAG), low-rank adaptation (LoRA), and log-structured merge (LSM) trees, and present the analogy of LLMs as lossy compressed databases.

2.1 Long-Term Context in LLMs

Maintaining long-term context in large language models is challenging due to fixed context window sizes. Standard transformers can only handle a limited number of tokens per query, which forces systems to either re-send the entire conversation history or risk losing important context. Approaches such as Transformer-XL introduce recurrence mechanisms to capture longer-term dependencies [3]. External memory systems and summarization techniques are used to compress prior interactions into manageable representations [2, 15]. Recent work explores memory compression and k-nearest neighbor mechanisms to retain important tokens, mitigating context loss from window limitations [22]. However, these methods involve trade-offs in complexity and accuracy, motivating our explicit state management.

2.2 Retrieval-Augmented Generation

RAG enhances LLMs by supplementing their parametric knowledge with an external, non-parametric memory. In RAG, relevant documents or facts are retrieved from a knowledge base and conditioned on by the model during generation [8, 13]. This dual memory system improves factual accuracy and allows for dynamic knowledge updates. Recent work has extended the RAG paradigm by integrating structured data sources to refine retrieval and reasoning. Recently, knowledge graphs have been proposed for RAG to enable structured relationships and multi-hop retrieval [7, 9]. However, RAG remains dependent on the quality of its retrieval mechanism and the underlying index, motivating our design wherein RAG serves as the immutable storage layer while offloaded data is managed through specialized adapters.

2.3 Updateable Memory and Knowledge Editing

Recent work has explored direct in-model knowledge editing techniques that enable targeted modifications of a model's internal representations without full retraining [6, 23]. These methods allow models to correct factual errors or incorporate new information with minimal disruption to their existing knowledge. In parallel, researchers have revisited differentiable memory architectures, such as Neural Turing Machines [5] and Memory Networks [21], to integrate persistent, updateable state into large language models. Although scaling these architectures to current model sizes remains challenging, they offer valuable insights into designing systems that continuously adapt and refine their stored knowledge.

2.4 Low-Rank Adaptation

LoRA offers an efficient method for adapting large language models by injecting small trainable low-rank matrices into the network while keeping the base model weights fixed [4,

10]. This approach dramatically reduces the number of parameters that need to be updated and the associated computational cost, making it an ideal mechanism for incorporating persistent state into LLMs. Unlike fine-tuning the entire model, which introduces significant overhead and reduces flexibility, LoRA provides a lightweight and modular alternative. Adapters can be easily updated or deleted without affecting the base model, whereas continuously updating all model weights would be computationally expensive and cumbersome to manage. Recent work has begun exploring using LoRA adapters for capturing and maintaining persistent, user-specific knowledge [11]. While the mathematical details of LoRA are beyond the scope of this workshop paper and not required for comprehension, its practical utility lies in enabling fine-tuning in scenarios where full model updates are impractical. Our framework leverages LoRA adapters to encode semantically grouped, persistent information, enabling efficient and modular updates without the overhead of full model retraining.

2.5 Log-Structured Merge Trees

LSM trees are a well-established data structure to optimize write-intensive workloads by maintaining a hierarchical storage system [14, 17]. In an LSM tree, recent data is stored in an in-memory component (akin to a memtable) and periodically flushed to disk as sorted SSTables. This hierarchical compaction process enables efficient updates while ensuring data integrity. We draw an analogy between LSM trees and the system proposed in this paper, notably with respect to how contextual knowledge is managed, flushed, and compacted throughout the system's lifecycle.

2.6 Towards Stateful LLMs

Viewed as lossy compressed databases, LLMs capture vast amounts of knowledge but do so in a static and approximate manner [18, 19]. Without explicit mechanisms for updating their internal state and inserting new knowledge, LLMs become outdated and struggle with maintaining extensive contexts over time.

The framework presented in this paper aims to bridge this gap by integrating RAG, LoRA, and hierarchical storage mechanisms to provide a robust foundation for developing stateful LLMs. By addressing the limitations of existing approaches, our framework offers a scalable solution for managing dynamic, evolving state information in AI systems.

3 System Design

Our system efficiently integrates short-term contextual understanding with long-term knowledge retention, enabling stateful and personalized interactions. The architecture combines RAG with LoRA-based adapters, creating a hierarchical storage system where the recent state is retrieved from the RAG knowledge base, and the older state is encoded into semantically grouped LoRA adapters.

Figure 1 illustrates the system's architecture and the flow of a user query through its main components. When a user submits a query, such as "Suggest a hotel and activities for my upcoming trip, preferably near the Eiffel Tower," the system processes it through several steps. First, the query is decomposed into core requests, transient instructions, and knowledge records. The knowledge records are stored in the knowledge base, which periodically flushes and trains new adapters in the background (not shown in the figure). The system then retrieves relevant knowledge records from the knowledge base and dynamically loads semantically similar adapters. Combined with the query and transient data, these components are used to construct an augmented prompt, while the LoRA adapters dynamically augment the LLM to generate the response.

This design ensures efficient retrieval of relevant knowledge, dynamic adapter loading, and the generation of tailored responses. The system maintains scalability and performance by periodically compacting and offloading knowledge into adapters while continuously updating its state based on past interactions. Although the adapters in the figure are labeled with broad categories (e.g., Dining or Cultural Activities), in practice, such groupings emerge from automatic clustering based on semantic vectors. To reduce clutter in Figure 1, offloaded records are omitted, and detailed knowledge records are summarized within the adapters.

To illustrate the system's stateful design, we assume the user had the following prior interactions before issuing the Paris travel query:

- "What are some good museums to visit in Rome?"
- "Find me a food tour in Florence with wine tasting."
- "I prefer staying in 4-star hotels."
- "What are some Michelin-starred restaurants in NYC?"
- "Plan a weekend in Barcelona focused on art and food."

These interactions seeded the knowledge base with persistent knowledge records and trained the LoRA adapters shown in Figure 1. The system leverages this accumulated knowledge in its responses to new queries.

3.1 Decomposition, Transformation, and Loading

The system decomposes each query into three components: the *core query*, which represents the main user request; *transient instructions*, which include query-specific details (e.g., "preferably near the Eiffel Tower") that do not persist; and *knowledge records*, which capture structured, reusable information (none in our running example). While not shown here, a possible knowledge record from the Paris travel query

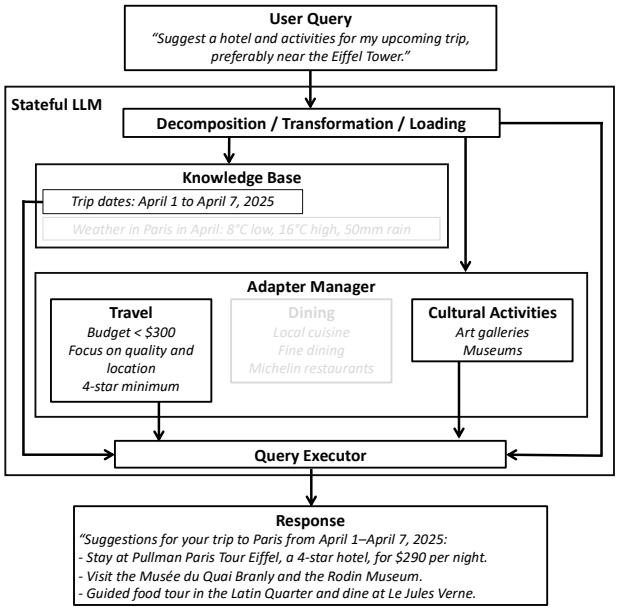


Figure 1: System architecture illustrating how a Paris travel query is processed. The system combines contextual knowledge records (e.g., trip dates with specialized adapters (e.g., Cultural Activities, Travel) to generate a personalized response.

could encourage the system to prefer hotels near landmarks in the future. After decomposition, the transient instructions and knowledge records are further transformed, and ambiguous references, such as "next Monday at 8," are disambiguated into precise formats to ensure consistent internal representation.

3.2 Knowledge Base for RAG

The Knowledge Base is the primary layer of persistent storage, where knowledge records are appended but never removed, ensuring an immutable history. It supports a RAG process, where queries trigger a semantic-retrieval step to fetch relevant key–value knowledge records. For example, the record "Trip dates -> April 1 to April 7, 2025" is retrieved based on its similarity to the query and passed to the query executor. While Figure 1 simplifies the representation by omitting offloaded knowledge records, all data reside in the Knowledge Base.

3.3 Adapter Manager

The second persistence layer consists of LoRA adapters, which dynamically fine-tune the base LLM without altering it. After new key-value knowledge records are added to the Knowledge Base, they are periodically offloaded into adapters, where each record is transformed into a fine-tuning sample (e.g., "Instruction: [key] Response: [value]"). Once an adapter has learned a knowledge record, it is marked as "offloaded," and the RAG process no longer retrieves it directly from the Knowledge Base. We combine the knowledge records used to train an adapter to create an embedding corresponding to that adapter's signature, which the system stores. For each query, the Adapter Manager evaluates adapters' signatures by semantic similarity to the query, dynamically loading the top K adapters with a similarity higher than threshold T_L to work alongside the base LLM in generating the response. We use a weighted combination of adapters based on similarity scores. In the example from Figure 1, the "Travel" and "Cultural Activities" adapters are selected to augment the LLM. An interesting property of LoRA adapters is that, unlike SStables in an LSM tree, they do not grow in size with additional data.

3.4 Query Executor

The Query Executor orchestrates the final response generation by combining multiple data sources: it retrieves relevant non-offloaded knowledge records from the Knowledge Base through semantic similarity (e.g., "Trip dates -> April 1 to April 7, 2025"), loads appropriate adapters selected by the Adapter Manager (such as "Travel" and "Cultural Activities"), and merges them with the base model. The Executor then constructs a final prompt by combining the user's core query with all contextual elements, passing this to the adapter-augmented LLM to generate a personalized response incorporating current and historical knowledge.

3.5 Flushing and Compaction

Our system draws inspiration from LSM trees to maintain efficiency through periodic background adapter management operations. When newly accumulated key-value knowledge records reach a fixed threshold, T_F (e.g., 100 records), they are flushed into new adapters.

Each adapter is associated with an embedding signature, and the system uses a fixed threshold, T_C, to determine when to compact adapters. If the similarity between two adapters exceeds T_C, the system retrieves the original knowledge records from the knowledge base, retrains a single consolidated adapter, and replaces the old ones. This approach prevents the proliferation of numerous small adapters while ensuring that knowledge remains cohesive and consistent.

Looking ahead, we plan to introduce additional triggers for compaction, such as detecting outdated or conflicting knowledge records to trigger compaction and clean up. Usage-based compaction, where frequently used adapters may be merged, and manual compaction triggers could also be incorporated. Currently, the system does not address scenarios

where compacting two adapters would exceed the maximum allowable number of knowledge records per adapter to prevent overwhelming a single adapter. Future enhancements could involve detecting when compaction would breach this limit and dynamically adjusting the similarity threshold to avoid merging the adapters.

4 Implementation

Our stateful LLM is implemented in Python as a dedicated class that encapsulates the base language model from the Transformers library (v4.48.3). It structures query inputs using a custom Python data class (comprising the query text, domain context, and persistent knowledge), manages token counts and context windows, and leverages Accelerate (v1.3.0) for distributed training while coordinating data retrieval from the Knowledge Base and the Adapter Manager for prompt assembly.

Knowledge Base Module. Knowledge records are stored as versioned JSON records with contextual tags and indexed using FAISS (v1.9.0) for efficient semantic retrieval; record embeddings are generated via Sentence-transformers (v3.4.1) based on a BERT model, with cosine similarity used to compare embeddings.

Adapter Manager Module. Developed with PyTorch and utilizing PEFT (v0.13.2) for LoRA adapters. We currently use $K = 4$ and fixed thresholds for adapter loading ($T_L = 0.7$), flushing ($T_F = 100$), and compaction ($T_C = 0.5$).

5 Evaluation

In this section, we conduct a comprehensive evaluation of our proposed stateful LLM framework. Our focus is on answering the following research questions:

RQ1: How does our stateful LLM solution, which integrates RAG and LoRA adapters, compare with a full-context LLM and a RAG-based LLM in terms of state retention and response accuracy? (§5.2)

RQ2: What are our state management approach's computational overheads and efficiency benefits, particularly regarding the cost of training and merging LoRA adapters versus traditional context handling? (§5.3)

5.1 Experimental Setup

Datasets. In our experiments, we utilize two sufficiently large real-world datasets to effectively stress-test our system. The first dataset is AG News [1], which comprises 120k news articles categorized into different topics. The second is 20 Newsgroups [12], a collection of approximately 20k newsgroup entries from 20 different newsgroups. We adapt these datasets for our purposes by manually augmenting them with questions to obtain question-answering datasets with extensive contexts.

Baselines. We compare against two baseline systems:

- **LLM with Full Context (LLM+C):** A model that processes queries by directly incorporating the entire context into the LLM's input without external state management.
- **LLM with RAG (LLM+RAG):** A retrieval-augmented generation system that manages persistent state via an external knowledge base.

Metrics. We use the following metrics:

- **BERT Score:** Measures the semantic similarity between the response and the ground truth, providing a quantitative evaluation of response quality [24].
- **Knowledge Retention:** Evaluates the fraction of facts accurately recalled and incorporated into the response, reflecting the system's ability to retain and retrieve persistent context.
- **Time:** Includes both training and inference time, depending on the context, to assess the computational efficiency of the system, including the cost of training and merging LoRA adapters.

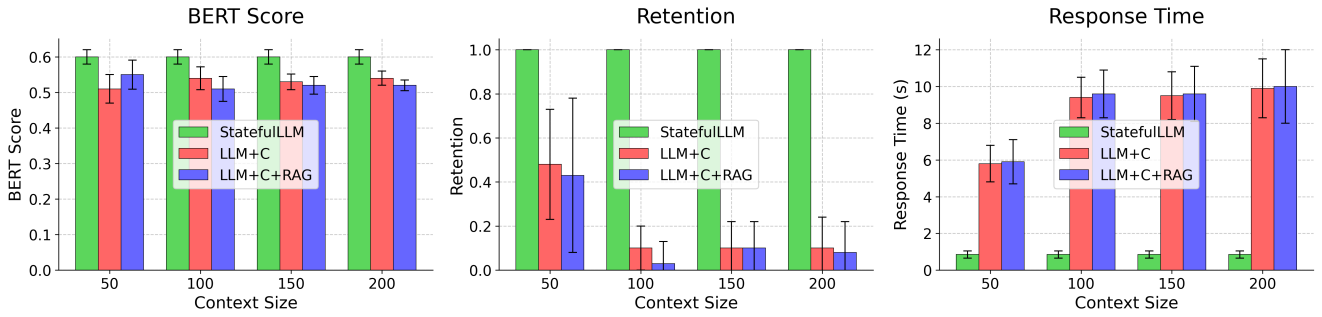
Models. We conduct our experiments using the Llama family of models [20], specifically `llama-2-7b-chat-hf`.

Hardware & Configuration. We use a machine running Debian Ubuntu and equipped with two H100 GPUs.

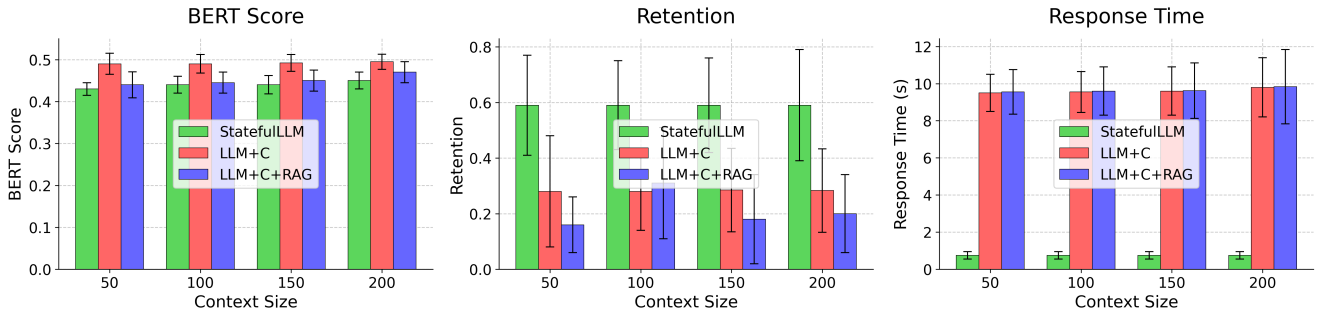
5.2 Overall Performance

In this set of experiments, we aim to answer RQ1 by evaluating the overall performance of our system across two datasets: AG News and 20 Newsgroups. We replay a subset of prompts in the dataset and collect the responses. Our objective is to assess how effectively our stateful LLM framework maintains persistent state and generates accurate responses. We also measure the average response for each system. We compare our stateful LLM framework with our two baselines, LLM with Full Context (LLM+C) and LLM with RAG (LLM+RAG). We show the results in Figure 2.

Overall, our stateful LLM approach outperforms both the full-context and RAG baselines in knowledge retention. It also executes significantly faster as, despite the overheads introduced by our framework, we drastically reduce the context size with LoRA adapters. The BERT score is lower for stateful LLM in the second dataset due to a cutoff in the maximum number of tokens. This last result warrants further future investigation as it indicates that the stateful LLM approach tends to generate longer responses.



(a) AG News.



(b) 20 Newsgroups.

Figure 2: Comparison between our Stateful LLM, an LLM using only context (LLM+C), and an LLM using context and RAG (LLM+RAG) for different datasets. The x axis shows context sizes from 50 to 200 knowledge records.

5.3 Microbenchmark

To address RQ2, we conduct a microbenchmark to quantify the trade-offs between the overhead introduced by state management and the efficiency gains achieved through our approach. These experiments focus on three aspects: the inference time when using a full LLM context versus dynamically loading knowledge records and adapters (response time), the computational cost of training LoRA adapters compared to maintaining a full context (flushing time), and the time required for adapter compaction (compaction time). We use the AG News dataset in this experiment. We present the results of this microbenchmark in Table 1.

Operation	Stateful LLM	LLM+C
Response Time	0.8 s	9.5 s
Flushing Time	65 s	N/A
Compaction Time	86 s	N/A

Table 1: Microbenchmark showing the time of different operations for our Stateful LLM and LLM+C.

These results show that, while a modest cost is associated with training and compacting adapters, these overheads are offset by a shorter inference time. While LoRA adapters introduce a marginal computation overhead, they compensate for it by using parametric memory, drastically reducing the context size and improving inference time when the context is large, such as in this experiment.

6 Discussion and Future Directions

Applications and Use Cases. While much of our discussion has focused on personalized contexts, the applicability extends well beyond user preference storage. Incrementally updating adapters based on newly ingested data is relevant for organizational knowledge bases, task automation agents that evolve with experience, or scenarios where large volumes of streaming data must be distilled into actionable insights. These adapters effectively become layered "memories" for the LLM, enabling it to serve as a continuously learning system. For example, an enterprise environment could use adapters to accumulate industry-specific knowledge over time, and an autonomous planning agent could rely on adapters to recall operational context in disaster relief or strategic settings.

Generalizability. While our framework shows promising results in two datasets, its generalizability to diverse workloads remains uncertain. Many real-world applications may involve unpredictable queries, rapidly evolving knowledge, or highly specialized contexts that may challenge the scalability and adaptability of our approach. Addressing these challenges and evaluating the framework across diverse scenarios is an important direction for future work.

Larger Models. Our framework's scalability becomes increasingly advantageous as the size of the model grows. LoRA adapters offer substantially better scaling properties than full-context approaches because they modify only a tiny fraction of parameters (typically $< 1\%$). Although inference still requires the full model, the training time scales primarily with the adapter size rather than with the base model dimensions. Critically, the rank parameter (r) in LoRA – which determines adapter complexity – can remain constant regardless of the base model size, making relative efficiency gains even greater for larger models, where full fine-tuning would be prohibitively expensive. As models scale, context window limitations become more restrictive relative to the model's capacity, further highlighting our system's advantages. The compression of knowledge into parameter space rather than token space provides an efficient solution to the quadratic attention complexity problem that plagues large-context designs.

Adapter Recall and Context Pinning. A key question arises as to whether LoRA adapters can reliably recall all the information on which they have been trained. Since the LoRA mechanism is inherently lossy, specific details may not be perfectly retained, leading to possible omissions. This challenge mirrors similar recall shortcomings in RAG, where knowledge retrieval depends on semantic alignment with the query. In practice, some knowledge records or user preferences may be so critical that they must never be dropped, and the requirements of specific critical applications may not be compatible with our approach. One solution is to explicitly pin these critical records to always be included in the final context rather than relying solely on an adapter's learned parameters. Nonetheless, these approaches do not fully guarantee incorporation into the final response, given that an LLM may sometimes overlook or misconstrue even pinned information.

Dynamic Thresholds and Multi-Level Retrieval. Related to recall is the question of how thresholds for selecting adapters and knowledge records should adapt. Currently, adapter selection is based on static similarity, but in dynamic applications, adjusting thresholds based on context, user-defined preferences, or computation/memory budget can improve retrieval precision and results. In more complex workflows, adapters may generate additional context that is itself used to trigger supplementary knowledge retrieval or adapter loading, forming a multi-level retrieval pipeline. Determining how far such recursive expansion should go, especially under tight latency or cost constraints, is an open research challenge.

Potential for Context Generation. A promising future direction involves treating adapters not only as parameter-space storage but also as mini compressed databases that can output relevant knowledge pieces directly. Rather than merging adapters into the base LLM for inference, these adapters could themselves generate textual summaries or expansions, which would then be fused into the final context. This dual use as a learned memory layer and a content generator could help integrate distributed knowledge across adapters. However, designing the control logic, ensuring consistency across outputs, and preventing redundancy remain significant challenges.

Privacy and Knowledge Deletion. Our system's design raises legitimate questions about privacy and data governance. Fine-tuning adapters to encode sensitive knowledge can lead to unwanted data retention, making it non-trivial to fully "delete" knowledge once learned. Methods for selectively removing records or for marking data as non-trainable would be essential for real-world adoption in privacy-sensitive environments. Adding a robust privacy-preserving layer to the adapter training and compaction processes is a major avenue of future work.

Outlook. The proposed system unlocks a broad range of possibilities for making LLM-based solutions more adaptive and context-aware. Many challenges remain, including ensuring precise recall, managing privacy, handling large-scale compaction without compromising performance, and optimizing adapter merging strategies to balance efficiency with rapid updates. In future work, we intend to refine multi-level retrieval pipelines, develop strategies for selective data retention and removal, and explore adapter-generated contexts for richer integrations. Through these advancements, we aim to realize the vision of an infinitely updatable, stateful AI framework that efficiently and responsibly accommodates both personal and organizational needs.

7 Conclusion and Future Work

In this paper, we present a novel framework for transforming stateless LLMs into efficient, stateful systems. By integrating RAG with LoRA techniques, our approach enables LLMs to maintain and update persistent knowledge in a hierarchical fashion, drawing inspiration from log-structured merge trees. The evaluation demonstrates that our stateful LLM framework achieves superior state retention and response

accuracy compared to traditional full-context and RAG-based methods while incurring minimal computational overhead.

Although preliminary, we believe this work establishes a foundation for stateful, infinitely updatable LLMs, paving the way for truly adaptive and personalized AI systems.

Acknowledgments

We would like to thank our anonymous reviewers as well as Chongyang Xu, Tejas Harith, and Saumya Chaturvedi for their help and feedback that improved this work.

References

[1] 2025. http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

[2] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2021. Longformer: The Long-Document Transformer. *arXiv preprint arXiv:2004.05150* (2021).

[3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2978–2988.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems* 36 (2024).

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *arXiv preprint arXiv:1410.5401* (2014).

[6] Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. 2024. Model editing harms general abilities of large language models: Regularization to the rescue. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 16801–16819.

[7] Maria Gutierrez, Kevin Lee, and Ling Zhao. 2024. HippoRAG: Hierarchical Knowledge Graphs for Enhanced Long-Term Context in Retrieval-Augmented Generation. In *Proceedings of the 2024 Conference on Neural Information Processing Systems*.

[8] Kelvin Guu, Kenton Lee, Michael Tung, Panupong Pasupat, and Wentau Chang. 2020. REALM: Retrieval-Augmented Language Model Pre-Training. In *International Conference on Machine Learning*. doi:10.48550/ARXIV.2002.08909

[9] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309* (2024).

[10] Edward J Hu, Yelong Shen, Philip Wallis, Zeyuan Allen-Zhu, Yuxin Li, and Liang Wang. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*. doi:10.48550/ARXIV.2106.09685

[11] Alex Kai, Ming Chen, and Ravi Patel. 2024. MiLP: Modular Low-Rank Personalization for Persistent Memory in Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

[12] Ken Lang. 1995. NewsWeeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*. 331–339.

[13] Patrick Lewis, Ethan Perez, Adriano Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Myle Lewis, Luke Zettlemoyer, and Sebastian Riedel. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*. doi:10.48550/ARXIV.2005.11401

[14] Cheng Luo and Michael J Carey. 2019. LSM-based Storage Techniques: A Survey. *Comput. Surveys* 52, 3 (2019), 1–34. doi:10.1145/3323215

[15] Juan Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On Faithfulness and Factuality in Abstractive Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 1906–1919.

[16] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. 2021. Fast model editing at scale. *arXiv preprint arXiv:2110.11309* (2021).

[17] Patrick O'Neil, Edward Cheng, Don Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). In *Acta Informatica*, Vol. 33. Springer, 351–385. doi:10.1007/s002360050048

[18] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. 2020. KILT: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252* (2020).

[19] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Alexander Bakhtin, Yuxiang Wu, and Noah Miller. 2019. Language Models as Knowledge Bases?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 246–257.

[20] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[21] Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory Networks. *arXiv preprint arXiv:1410.3916* (2014).

[22] Felix Wu et al. 2022. Memorizing Transformers. In *Proceedings of the 2022 International Conference on Machine Learning*.

[23] Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172* (2023).

[24] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SkeHuCVFDr

Received 24 February 2025; revised 11 March 2025; accepted 3 March 2025