From Isolated Conversations to Hierarchical Schemas: Dynamic Tree Memory Representation for LLMs

Alireza Rezazadeh **Zichao Li** **Wei Wei** **Yujia Bao**

Center for Advanced AI, Accenture
{alireza.rezazadeh,zichao.li,wei.h.wei,yujia.bao}@accenture.com

Abstract

Recent advancements in large language models have significantly improved their context windows, yet challenges in effective long-term memory management remain. We introduce **MemTree**, an algorithm that leverages a dynamic, tree-structured memory representation to optimize the organization, retrieval, and integration of information, akin to human cognitive schemas. MemTree organizes memory hierarchically, with each node encapsulating aggregated textual content, corresponding semantic embeddings, and varying abstraction levels across the tree's depths. Our algorithm dynamically adapts this memory structure by computing and comparing semantic embeddings of new and existing information to enrich the model's context-awareness. This approach allows MemTree to handle complex reasoning and extended interactions more effectively than traditional memory augmentation methods, which often rely on flat lookup tables. Evaluations on benchmarks such as the Multi-Session Chat (MSC) and MultiHop RAG show that MemTree significantly enhances performance in scenarios that demand structured memory management.

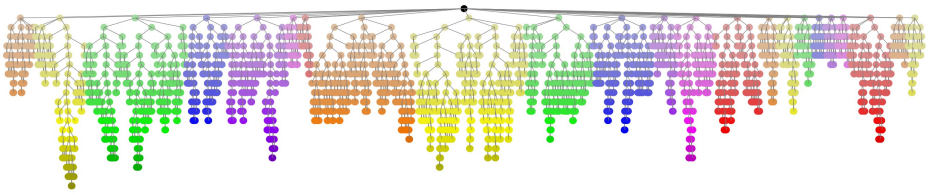

Figure 1: **MemTree** (subset) developed on the MultiHop RAG [18]. MemTree updates its structured knowledge when new information arrives, enhancing inference-time reasoning capabilities of LLMs.

1 Introduction

Despite recent advances in large language models (LLMs) where the context window has expanded to millions of tokens [7, 6, 5], these models continue to struggle with reasoning over long-term memory [11]. This challenge arises because LLMs rely primarily on a key-value (KV) cache of past interactions, processed through a fixed number of transformer layers, which lack the capacity to effectively aggregate extensive historical data. Unlike LLMs, the human brain employs dynamic memory structures known as schemas, which facilitate the efficient organization, retrieval, and integration of information as new experiences occur [2, 9]. This dynamic restructuring of memory is a cornerstone of human cognition, allowing for the flexible application of accumulated knowledge across varied contexts.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

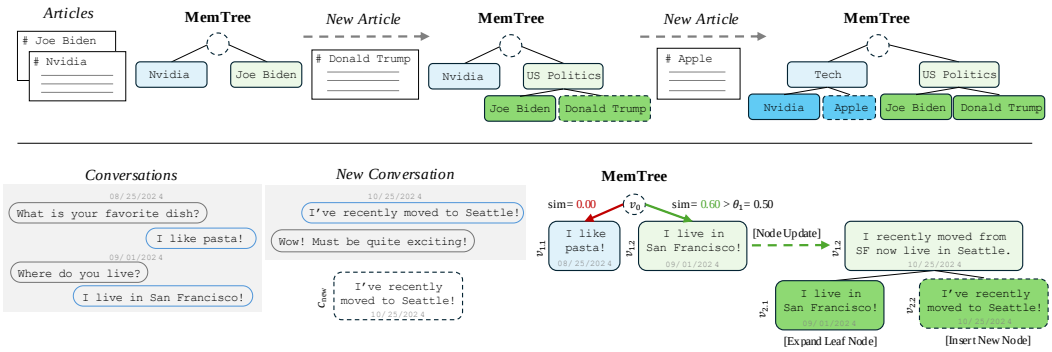

Figure 2: Illustration of MemTree. MemTree represents knowledge schema via a dynamic tree. Both parent and leaf nodes archive textual content, summarizing information relevant to their respective levels. Upon receiving new information, the system begins traversal from the root node. If the new information is semantically akin to an existing leaf node under the current node, it is routed to that node. Conversely, if it diverges from all existing leaf nodes under the current node, a new leaf node is created under the current node, concluding the traversal. During this process, all ancestor nodes will integrate the new information into the higher-level summaries they maintain.

A prevalent method to address the limitations of long-term memory in LLMs involves the use of external memory. [20] introduced the concept of utilizing external memory for the storage and retrieval of relevant information. More recent approaches in LLM research have explored various techniques to manage historical observations in databases, retrieving pertinent data for given queries through vector similarity searches in the embedding space [15, 14, 22]. However, these methods primarily utilize a lookup table for memory representation, which fails to capture the inherent structure in data. Consequently, each past experience is stored as an isolated instance, lacking the interconnectedness and integrative capabilities of the human brain's schemas. This limitation becomes increasingly problematic as the size of the memory grows or when relevant information is distributed across multiple instances.

In this work, we introduce **MemTree**, an algorithm that mimics the schema-like structures of the human brain by maintaining a dynamically structured memory representation during interactions. Within MemTree, each memory unit is represented as a node within a tree, containing node-level information and links to child nodes.

Upon encountering new information, MemTree updates its memory structure starting from the root node. It evaluates at each node whether to instantiate a new child node or integrate the information into an existing child node. This decision process is governed by a traversal algorithm that efficiently adds new information with an insertion complexity of O(log N), where N denotes the number of conversational interactions. This structure facilitates the aggregation of knowledge at parent nodes, which evolve to capture high-level semantics as the tree expands. For knowledge retrieval, MemTree computes the cosine similarity between node embeddings and the query embedding. This method maintains the retrieval time complexity comparable to existing approaches, ensuring efficiency.

We assessed the performance of MemTree, using two benchmarks: Multi-Session Chat (MSC) [14] and MultiHop RAG [18]. In MSC, we explored both the standard format, which encompasses 15 conversation turns, and a modified long-context format that extends these conversations to 200 turns. We observed that while initially incorporating chat histories directly into GPT-4o shows promising performance, it deteriorates as the number of conversation rounds increases. In contrast, MemTree consistently maintains high performance as conversations progress, outperforming other memory approaches such as MemoryStream [15] and MemGPT [14]. In MultiHop RAG, MemTree demonstrates further performance advantages on queries that require comparative analysis and temporal reasoning over the chat history. Additionally, despite being an online algorithm, MemTree closely matches the performance of the offline algorithm RAPTOR [17], which relies on hierarchical clustering and complete data access to build a structured knowledge base. Our qualitative analyses, including visualizations of the learned tree structures, reveal that MemTree meaningfully captures the semantics of the context.

2 Related Work

Recent large language models (LLMs), such as GPT-4 [1], PaLM [3], and LLaMA [19], excel in various natural language processing tasks but struggle with long-term memory and retrieving information from past interactions. Researchers have explored leveraging external memory for long-range reasoning in traditional RNNs [20, 16, 10, 12]. Building on these concepts, recent methods aim to augment LLMs with enhanced memory capabilities.

Memory-Augmented LLMs [22] introduced MemoryBank, a long-term memory framework that stores timestamped user dialogues and incorporates an exponential decay mechanism to gradually forget outdated information. MemGPT [14] proposed automatic memory management by leveraging LLM function-calling for tasks such as conversational agents and document analysis, providing a schema for accessing and modifying a memory database. Similarly, [15] developed LLM-based generative agents with memory streams that log experiences as textual descriptions with timestamps. Retrieval functions select memories by calculating a weighted score based on recency, importance, and relevance, with the latter two determined by prompting the LLM.

These approaches represent the most common solutions for enhancing LLMs with memory, relying primarily on storing and managing information in tabular databases and retrieving relevant instances through vector similarity search. However, as memory scales or information becomes dispersed across multiple entries, the limitations of these unstructured memory representations become apparent, leading to difficulties in effectively retrieving complex or scattered data.

Augmenting LLMs with Relation Triplets [13] proposed a triplet-based structured memory unit for LLMs, where each triplet encodes a relationship between two arguments. Similarly, [4] introduced a graph-based triplet representation for text-based games, where LLMs parse observations into object-relation-object triplets to construct a semantic graph memory. During retrieval, the most relevant triplets are selected via semantic search to support decision-making and planning. While these methods are effective for tasks like storing individual relations [13] or constructing scene graphs [4], triplet-based approaches operate at the object level, limiting their scalability to larger, more complex datasets. Additionally, a strictly triplet-based structure struggles to generalize to more generic information that doesn't fit neatly into a triplet relational format.

Structured RAG Approaches As retrieval-augmented generation (RAG) continues to gain attention, recent efforts have focused on enhancing traditional RAG approaches by incorporating structured knowledge representations. These methods improve the navigation and summarization of large text corpora, particularly in complex question-answering tasks. RAPTOR [17] organizes large text into a recursive tree structure by clustering and summarizing text chunks at multiple layers, enabling efficient retrieval that captures both high-level themes and detailed information. GraphRAG [8] constructs a knowledge graph from entities and relationships extracted using LLMs, partitioning the graph into modular communities. These communities are summarized independently, with responses combined using a map-reduce approach to answer global queries. While these structured retrieval methods show promise for large-scale text processing, they focus primarily on knowledge graphs and static corpora, leaving the challenge of online memory retrieval largely unaddressed.

3 Methods

MemTree represents memory as a tree $T = (V, E)$, where V is the set of nodes, and $E \subseteq V \times V$ is the set of directed edges representing parent-child relationships. Each node $v \in V$ is represented as:

$$v = [C_v, e_v, p_v, \mathcal{C}_v, d_v]$$

where:

- c_v: the textual content aggregated at node v.
- $e_v \in \mathbb{R}^d$: an embedding vector derived using an embedding model $f_{\text{emb}}(c_v)$.
- $p_v \in V$: the parent of node v.
- $\mathcal{C}_v \subseteq V$: the set of children of node v, with edges directed from v to each $u \in \mathcal{C}_v$.
- d_v: the depth of node v from the root node v_0.

Note that the root node v_0 serves as a structural node, containing neither content nor embedding, i.e., $c_{v_0} = \varnothing$ and $e_{v_0} = \varnothing$.

MemTree utilizes this tree-structured representation to dynamically track and update the knowledge exchanged between the user and the LLM. While less flexible than a generic graph architecture, the tree structure inherently biases the model towards hierarchical representation. Additionally, trees offer efficient complexity for insertion and traversal, making the algorithm suitable for real-time online use cases.

When new information (converstaion) is observed, MemTree dynamically adapts by traversing the existing structure, identifying the appropriate subtree for integration, and updating relevant nodes (Section 3.1). This process, illustrated in Figure 1, ensures the proper integration of new information while preserving the underlying context and hierarchical relationships within the memory. When retrieving information from the memory, MemTree simply compares the embeddings of the query message with the embeddings of each node in the tree, returning the most relevant nodes (Section 3.2).

3.1 Memory Update

The memory update procedure in MemTree is triggered upon observing new information (e.g., a new conversation). This procedure ensures that the tree structure dynamically adapts and integrates new data while maintaining a coherent hierarchical representation.

Attaching New Information by Traversing the Existing Tree To integrate new information, we begin by creating a new node v_{new} with the textual content $c_{v_{\text{new}}}$. Then we start tree traversal from the root node. At each node v, MemTree evaluates the *semantic similarity* between the new information $c_{v_{new}}$ and the children of the current node in the embedding space. This evaluation is performed by computing the embedding $e_{v_{\text{new}}} = f_{\text{emb}}(c_{v_{\text{new}}})$ for the new content $c_{v_{\text{new}}}$ and comparing it to the embeddings of the child nodes $\mathcal{C}(v)$ of the current node v using cosine similarity.

This similarity evaluation drives the following decisions:

- **Traverse Deeper:** If a child node's similarity exceeds a depth-adaptive threshold $\theta(d_v)$, traversal continues along that path. If multiple child nodes meet this criterion, the path with the highest similarity score is chosen.
  - **Boundary:** When traversal reaches a leaf node, the leaf is expanded to become a parent node, accommodating both the original leaf node and v_{new} as children. The parent's content is then updated to aggregate both the original leaf node's content and the new information $c_{v_{\text{new}}}$. The details of this aggregation process will be explained below.
- **Create New Leaf Node:** If all child nodes' similarities are below the threshold $\theta(d_v)$, v_{new} is directly attached as a new leaf node under the current node.

The similarity threshold $\theta(d)$ is adaptive based on the node's depth d, defined as:

$$\theta(d) = \theta_0 e^{\lambda d},$$

where θ_0 is the base threshold, and λ controls the rate of increase with depth. This mechanism ensures that deeper nodes, which represent more specific information, require a higher similarity for new data integration, thereby preserving the tree's hierarchical integrity. Further details, including specific parameter values, are provided in the Appendix A.1.3.

Updating Parent Nodes Along the Traversal Path Once v_{new} is inserted, the content and embeddings of all parent nodes v along the traversal path are updated to reflect the new information. This is achieved through a conditional aggregation function:

$$c'_v \leftarrow \text{Aggregate}(c_v, c_{\text{new}} \mid n),$$

where c'_v is the updated content, and $n = |\mathcal{C}(v)|$ is the number of descendants of node v. The aggregation function, implemented as an LLM-based operation, combines the existing content c_v with the new content c_{new}, conditioned on n. As n increases, the aggregation abstracts the content further to balance the existing and new information (see Appendix A.1.2 for more details).

The embedding of the parent node is then updated as:

$$e_v \leftarrow f_{\text{emb}}(c'_v),$$

ensuring that the parent node effectively represents both the new and existing information. This process maintains the hierarchical organization of the memory as the tree expands, enabling MemTree to adaptively and accurately represent the evolving conversation.

The complete memory update process is outlined in Algorithm 1.

3.2 Memory Retrieval

Efficient and effective retrieval of relevant information is crucial for ensuring that MemTree can provide meaningful responses based on past conversations. Inspired by RAPTOR [17], we adopt the collapsed tree retrieval method, which offers significant advantages over traditional tree traversal-based retrieval.

Collapsed Tree Retrieval The collapsed tree approach enhances the search process by treating all nodes in the tree as a single set. Instead of conducting a sequential, layer-by-layer traversal, this method flattens the hierarchical structure, allowing for simultaneous comparison of all nodes. This technique simplifies the retrieval process and ensures a more efficient search.

The retrieval process involves the following steps:

1. Query Embedding: Embed the query q using $f_{\text{emb}}(q)$ to obtain e_q.
2. Similarity Computation: Calculate cosine similarities between e_q and all tree nodes.
3. Filtering: Exclude nodes with similarity scores below a threshold θ_{retrieve}.
4. Top-K Selection: Sort the remaining nodes by similarity and select the top-k most relevant nodes.

4 Experiments

4.1 Datasets

We evaluate the effectiveness of MemTree across various settings using three datasets: Multi-Session Chat, Multi-Session Chat Extended and MultiHop RAG. These datasets were selected to represent different interactive contexts—dialogue interactions and information retrieval from multiple texts, respectively, providing a comprehensive test bed for our model.

- Multi-Session Chat (**MSC**): The dataset was introduced by [21]. In this work, we consider the processed version provided by [14]. The dataset consists of 500 sessions, each featuring approximately 15 rounds of synthetic dialogue between two agents. Each session includes follow-up questions that challenge the model to retrieve and utilize information from prior dialogues within the same session.
- Multi-Session Chat Extended (**MSC-E**): To test the performance for even longer conversation rounds, we expanded MSC by generating an additional 70 sessions, each containing about 200 rounds of dialogue. In these extended sessions, a follow-up question follows each conversation round, demanding more precise and timely information retrieval across the dialogues. We detail the extension methodology in Appendix A.3.
- MultiHop RAG: This dataset comprises 609 distinct news articles across six categories [18]. It includes 2,556 multi-hop questions requiring the integration of information from multiple articles to formulate comprehensive answers. We consider three question types: inference, comparison, and temporal reasoning, each adding a layer of complexity to the information retrieval process.

4.2 Baselines

We compare MemTree with three methods along with a naive baseline, which involves concatenating all chat histories and feeding them into a large language model (LLM):

- MemoryStream: This baseline, proposed by [15], employs a flat lookup-table style memory that logs chat histories through an embedding table. The primary distinction between MemTree and this baseline is that MemTree utilizes a structured tree representation for the memory and models high-level representations throughout the memory insertion process.

Table 1: Comparison of Naive Conversation History Combination vs. External Memory on the MSC Dataset. Due to the dataset's brevity (15 dialogue rounds with fewer than 1,000 tokens), naively concatenating the entire history and sending it to GPT-4o delivers the best performance. Among models that only send the query to GPT-4o, MemTree surpasses MemGPT and MemoryStream in both accuracy and ROUGE scores.

Model	**Context**	**Accuracy** ⇑	**ROUGE-L (R)** ⇑
Results reported by [14]			
GPT-4 Turbo	Query + Full history summary	35	35
GPT-4 Turbo	Query + Full history summary + MemGPT	93	82
Our results with GPT-4o and text-embedding-3-large			
GPT-4o	Query + Full history	**95.6**	**88.0**
GPT-4o	Query + MemGPT [14]	70.4	68.6
GPT-4o	Query + MemoryStream [15]	84.4	79.1
GPT-4o	Query + **MemTree (ours)**	84.8	79.9

- MemGPT: [14] introduces a memory system designed to update and retrieve information efficiently. It uses an OS paging algorithm to evict less relevant memory into external storage. However, like MemoryStream, it does not format high-level representations. For our experiments, we utilized the official MemGPT implementation available on GitHub (`https://github.com/cpacker/MemGPT`).
- RAPTOR: This method [17] constructs a structured knowledge base using hierarchical clustering over all available information. The key difference between MemTree and this baseline is that MemTree operates as an *online* algorithm, updating the tree memory representation on-the-fly based on incoming knowledge, while RAPTOR applies hierarchical clustering on a fixed dataset.

We employ OpenAI's `GPT-4o` (version 2024-05-13) and `text-embedding-3-large` for all baselines and MemTree. This standardization ensures that any observed performance differences are attributable to the memory management methodologies rather than variations in underlying model capabilities or embeddings.

4.3 Implementation Details and Evaluation Metrics

Following previous work [14, 18], we evaluate MemTree through end-to-end question answering performance. The experimental procedure for each query involves four steps:

1. Load the corresponding dialogue/history into the memory.
2. Retrieve the relevant information from the memory based on the given query.
3. Use GPT-4o to answer the query based on the retrieved information.
4. Evaluate the generated answer using one of the following two metrics: 1) Use GPT-4o to compare the generated answer with the reference answer, resulting in a binary accuracy score; 2) Evaluate the ROUGE-L recall (R) metric of the generated answer compared to the relatively short gold answer labels, without involving the LLM judge.

The detailed prompts for steps 3 and 4 can be found in Appendix A.2. Other implementation details for MemTree can be found in Appendix A.1.

5 Results

5.1 Multi-Session Chat

We present the MSC results in Table 1. For the naive baseline, directly providing the full history to GPT-4o yields the best result, achieving an accuracy of 96%. This outcome is expected, given that the entire dialogue consists of only 15 rounds and fewer than one thousand tokens. However, providing a summary of the chat history significantly drops performance to 35%, even for the more

Table 2: **Accuracy** on MSC-E. The MSC-E dataset extends the original MSC dataset from 15 dialogue rounds to 200, serving as a more suitable test bed for evaluating the capability of memory algorithms in reasoning over long-context histories. Both MemoryStream and MemTree outperform the naive full history baseline, underscoring the necessity of external memory for effective long-context reasoning. We present the overall accuracy and provide a detailed breakdown based on the positions (dialogue round index) of the supporting evidence. For standard deviations, refer to Figure 4.

Model	**Context**	**Position of the supporting evidence**					**Overall**
		0-40	*40-80*	*80-120*	*120-160*	*160-200*	
GPT-4o	Query + Full history	**84.5**	78.3	75.5	74.4	76.7	78.0
GPT-4o	Query + MemoryStream	78.5	81.0	81.0	81.4	81.8	80.7
GPT-4o	Query + **MemTree (ours)**	82.1	**82.1**	**82.3**	**82.3**	**84.2**	**82.5**

Table 3: **Accuracy** on MultiHop RAG. We report the results on three query types: (1) Inference queries, which require reasoning from retrieved information to provide answers; (2) Comparison queries, which involve evaluating evidence within the retrieved data to compare different entities or values; and (3) Temporal queries, which analyze time-related information to determine the sequence of events. MemTree outperforms MemoryStream by a large margin on both comparison and temporal queries, closing the gap to the offline method RAPTOR.

Model	**Context**	**Inference**	**Comparison**	**Temporal**	**Overall**
GPT-4o	Human Annotated Evidence	98.4	80.1	55.5	79.2
Offline method					
GPT-4o	RAPTOR	96.5	76.5	66.0	81.0
Online method					
GPT-4o	MemoryStream	**96.0**	64.8	59.3	74.7
GPT-4o	**MemTree (ours)**	95.0	**74.1**	**65.3**	**79.4**

powerful GPT-4 Turbo model [14]. This decline occurs because the summary may not cover the topics the query is addressing. To directly compare the performance of different memory management algorithms, we consider the setting where only the query and the retrieved information are provided to the LLM. In this scenario, MemTree surpasses both MemStream and MemGPT.[1]

5.2 Multi-Session Chat Extended

Table 2 presents the results on MSC-E. We observe that both MemoryStream and MemTree achieve better overall accuracy than the naive baseline, which directly uses the full history. This illustrates the importance of having an external memory system as the conversation history grows. When we break down the accuracies based on the positions of the supporting evidence within the entire dialogue, we find that the naive baseline performs best when the evidence is presented early on, likely due to position bias [11]. It is worth noting that since MemTree updates the memory sequentially based on the order of the dialogue, it inherently favors more recent conversations over older ones. This bias is demonstrated in Table 2, where the accuracy increases from 82.1 to 84.2. Nevertheless, MemTree consistently outperforms MemoryStream across all positions (see Figure 4 for a visualization).

5.3 MultiHop RAG

Table 3 summarizes the end-to-end performance of MultiHop RAG with different memory retrieval algorithms. All methods perform exceptionally well on inference-style questions, which primarily focus on fact-checking based on a document, achieving over 95% accuracy. MemTree slightly underperforms compared to MemoryStream in this category by less than one percentage point. However, when evaluating performance on questions that require the comparison of different documents or

[1]The MemGPT GitHub codebase has been modified, and we were unable to reproduce the results.

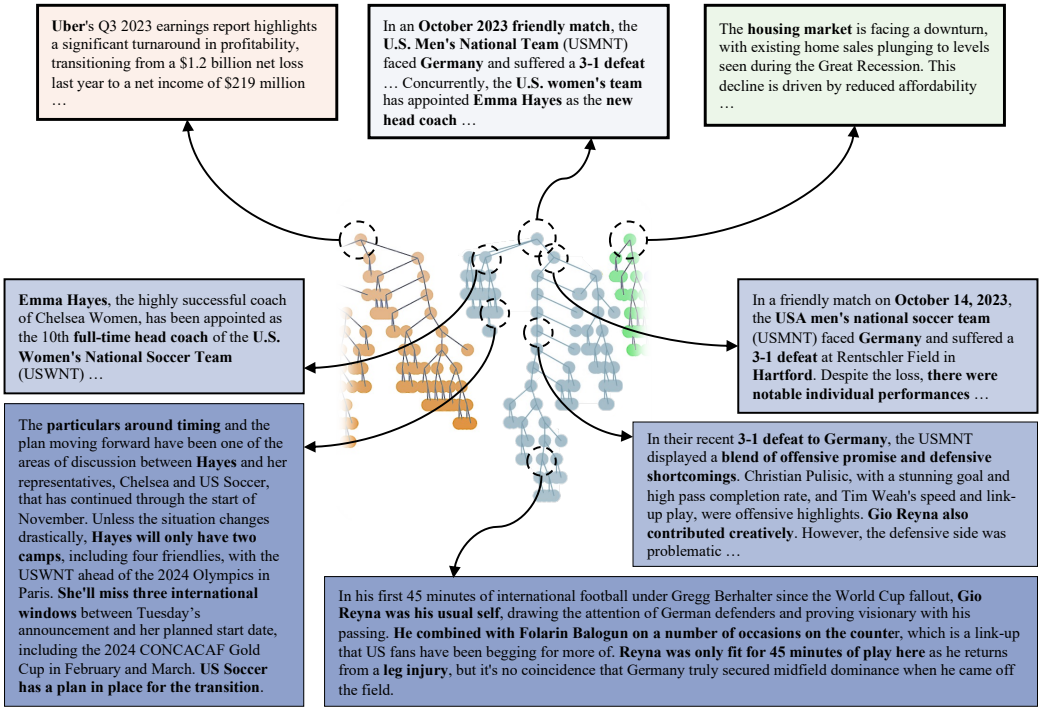

Figure 3: Visualization of the Learned MemTree Structure on the MultiHop RAG Dataset. Due to space limitations, we display only a small subtree from the entire tree (a larger subtree is depicted in Figure 1). As we traverse deeper into the tree, the content stored in the nodes becomes increasingly specific. For instance, the three blue nodes shown in the bottom right corner begin with a general summary of the *USMNT's 3-1 defeat to Germany*, then branch into *specific insights on individual performances and team dynamics*, and ultimately delve into *a detailed analysis of Gio Reyna's impact during the match*. Note that all intermediate contents in the parent nodes are generated by MemTree during the node update step. This hierarchical organization demonstrates how MemTree efficiently stores and retrieves information, progressing from overarching concepts to specific details.

temporal reasoning, MemTree excels. It outperforms MemoryStream by 9.35 percentage points on comparison-style questions and by 6.01 percentage points on temporal-style questions. Notably, its performance is only slightly below the offline method RAPTOR, which assumes access to the full set of information, by 2 percentage points (see Figure 5 for a visualization of Table 3).

Another observation from the table is that while humans can annotate evidence fairly accurately for inference and comparison-style questions, the annotated evidence for temporal questions is less precise. This results in worse performance than the model-derived memory for temporal questions.

We also visualize the learned memory tree in Figure 1 and Figure 3. The hierarchical structure learned by the model corresponds to their semantic hierarchies, and the intermediate parent nodes effectively capture high-level information during the memory insertion process.

6 Conclusion

MemTree effectively addresses the long-term memory limitations of large language models by emulating the schema-like structures of the human brain through a dynamic tree-based memory representation. This approach enables efficient integration and retrieval of extensive historical data, as demonstrated by its superior performance on Multi-Session Chat (MSC) and MultiHop RAG benchmarks. Our evaluations reveal that MemTree consistently maintains high performance and demonstrates human-like knowledge aggregation by capturing the semantics of the context within its tree memory structure. This advancement offers a promising solution for enhancing the reasoning capabilities of LLMs in handling long-term memory.

Disclaimer

This content is provided for general information purposes and is not intended to be used in place of consultation with our professional advisors. This document refers to marks owned by third parties. All such third-party marks are the property of their respective owners. No sponsorship, endorsement or approval of this content by the owners of such marks is intended, expressed or implied.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] John R Anderson. *Cognitive psychology and its implications*. Macmillan, 2005.

[3] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[4] Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*, 2024.

[5] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[6] Aydar Bulatov, Yuri Kuratov, Yermek Kapushev, and Mikhail S Burtsev. Scaling transformer to 1m tokens and beyond with rmt. *arXiv preprint arXiv:2304.11062*, 2023.

[7] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*, 2024.

[8] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[9] Vanessa E Ghosh and Asaf Gilboa. What is a memory schema? a historical perspective on current neuroscience literature. *Neuropsychologia*, 53:104–114, 2014.

[10] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387. PMLR, 2016.

[11] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[12] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

[13] Ali Modarressi, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Ret-llm: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*, 2023.

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

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

[16] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[17] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*, 2024.

[18] Yixuan Tang and Yi Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*, 2024.

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

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

[21] J Xu. Beyond goldfish memory: Long-term open-domain conversation. *arXiv preprint arXiv:2107.07567*, 2021.

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

A Appendix

A.1 MemTree Details

Further details and parameter settings for our approach are outlined below. Unless otherwise specified, these settings are consistent across all experiments presented in the paper.

A.1.1 MemTree Algorithm

The following outlines the algorithmic procedure for incrementally updating and restructuring the memory representation in MemTree. This approach ensures that new information is efficiently integrated into the existing memory hierarchy while dynamically adjusting based on content similarity and structural depth.

Parameters:

- c: the textual content stored at a node or introduced as new information.
- e: the embedding vector representing the content, generated by an embedding function f_{emb}.
- v: a node in the memory tree, which contains content, embeddings, and connections to other nodes. Note that the root is a structural node and does not hold content.
- d: the depth of a node in the tree.

Algorithm 1 Adding New Information to MemTree

Require: New information c_{new}, root node v_0, threshold function $\theta(d)$
$e_{\text{new}} \leftarrow f_{\text{emb}}(c_{\text{new}})$
INSERTNODE(v_0, e_{new}, c_{new}, 0)
procedure INSERTNODE(v, e_{new}, c_{new}, d)
    if v is a leaf **then**
        Expand v into a parent
        Create and attach child node v_{leaf} with original content
    end if
    Compute similarity $s_i = \text{sim}(e_{\text{new}}, e_i)$ for each child v_i of v
    $v_{\text{best}} \leftarrow \arg\max(s_i)$, $s_{\text{max}} \leftarrow \max(s_i)$
    if $s_{\text{max}} \geq \theta(d)$ **then**
        $c_v \leftarrow \text{Aggregate}(c_v, c_{\text{new}})$
        $e_v \leftarrow f_{\text{emb}}(c_v)$
        INSERTNODE(v_{best}, e_{new}, c_{new}, $d+1$)
    else
        Create and attach new child node v_{child} with c_{new}
    end if
end procedure

A.1.2 Aggregate Operation

When new information is added, the content of parent nodes along the traversal path is updated through a conditional aggregation. This process combines the existing content of the parent node with the new content, factoring in the number of its descendants. The aggregation operation is implemented using the following prompt:

```
You will receive two pieces of information: New Information is
detailed, and Existing Information is a summary from {n_children}
previous entries. Your task is to merge these into a single,
cohesive summary that highlights the most important insights.
- Focus on the key points from both inputs.
- Ensure the final summary combines the insights from both pieces of
information.
- If the number of previous entries in Existing Information is
accumulating (more than 2), focus on summarizing more concisely,
```

```
only capturing the overarching theme, and getting more abstract in
your summary.
Output the summary directly.
[New Information]
{new_content}
[Existing Information (from {n_children} previous entries)]
{current_content}
[ Output Summary ]
```

A.1.3 Adaptive Similarity Threshold

The adaptive similarity threshold ensures that deeper nodes, representing more specific information, require higher similarity for new data integration, while shallower nodes are more abstract and accept broader content. This mechanism preserves the tree's hierarchical integrity by adjusting selectivity based on the node's depth. The threshold is computed as:

$$\text{threshold} = \text{base\_threshold} \times \exp\left(\frac{\text{rate} \times \text{current\_depth}}{\text{max\_depth}}\right)$$

where:

- base_threshold $= 0.4$
- rate $= 0.5$
- current_depth is the depth of the current node.
- max_depth is the maximum depth of the tree.

A.1.4 Retrieval

For the MSC experiment, the retrieval system returns the top $k = 3$ similar dialogues from 15-round conversations, with a context length of 1000 tokens for all models. In the MSC-E dataset, due to longer conversations, the retrieval returns the top $k = 10$ similar dialogues, with a context length of 8192 tokens to accommodate the models with full-chat history. This setting is similarly applied to the Multihop RAG experiment, where longer contexts are required.

A.2 Further Experimental Detials

A.2.1 Evaluation Metrics

Predicted Response Generation: To assess retrieval performance, we configure the LLM to generate a response to the query based solely on the retrieved content using the following prompt:

```
Write a high-quality short answer for the given question using only
the provided search results (some of which might be irrelevant).
[ Question ]
{query}
[ Search Results ]
{retrieved_content}
[ Output ]
```

Binary Accuracy Evaluation: To measure binary accuracy across all experiments, we employed the following prompt, instructing the model to evaluate the predicted response against the ground-truth answer:

```
Your task is to check if the predicted answer appropriately responds
to the query in a similar way as the ground-truth answer.
Instructions:
- Output '1' if the predicted answer addresses the query similarly
to the ground-truth answer.  - Output '0' if it does not.  - Only
output either '0' or '1'.  No explanations or extra text.
```

```
[ Query ]
{query}
[ Ground-Truth Answer ]
{gt_answer}
[ Predicted Answer ]
{predicted_answer}
[ Output ]
```

A.3 MSC-E Data Generation

Building on the MSC dataset from [14], we extend each conversation to 200 rounds using the following iterative process. A sliding window of the most recent 8 turns is maintained, and for each step, the next 2 rounds of dialogue are generated using the prompt below. This approach allows for a natural progression of conversation while keeping the context manageable for the model:

```
Generate a continuation of the conversation between Alex and Bob.
Follow these guidelines:

 1. Alternate strictly between Alex and Bob, starting with Alex.
 2. Alex should speak exactly {n_rounds} times, and Bob should speak
    exactly {n_rounds} times.
 3. Each turn should consist of 1-3 sentences.
 4. Ensure that each response flows logically and organically from
    the previous turn, avoiding forced transitions or unnatural
    questions.
 5. Focus on developing rapport between the characters.  Use a mix
    of statements, reactions, and occasional questions to maintain a
    conversational tone.
 6. Allow the conversation to transition smoothly between topics,
    keeping it casual and coherent.

[ Conversation History ]
{recent_chat_hist}
[ Generated Dialogue ]
```

Output Example: Below is an excerpt from the MSC-E dataset, showcasing one session of a conversation that spans 200 rounds in total.

```
Alex:  Hi!  How are you doing tonight?
Bob:  I'm doing great.  Just relaxing with my two dogs.
Alex:  Great.  In my spare time I do volunteer work.
Bob:  That's neat.  What kind of volunteer work do you do?

...

Alex:  That would be great!  I'd love to try some of your Thai
recipes.  Cooking can be such a creative outlet, don't you think?
Bob:  Absolutely, it's like a culinary adventure in your own kitchen.
Speaking of adventures, have you planned any trips lately, maybe to
explore new cuisines firsthand?
Alex:  Not yet, but I've been dreaming of a trip to Italy to indulge
in the food and scenery.  How about you, any travel plans on the
horizon?
Bob:  I've been thinking about visiting Japan.  I'm fascinated by
their culture and, of course, the sushi!  It would be an amazing
experience to see it all in person.
Alex:  Japan sounds incredible!  The blend of traditional and modern
aspects in their culture is so intriguing.  You'll have to share
your experiences if you go.
```

A.4 MSC-E Query Generation

To generate queries and ground-truth responses for evaluating memory retrieval quality, we apply the following prompt to subsets of the conversation history. The generated questions will help assess how effectively the memory captures and retrieves information from various points in the dialogue:

```
Based on the conversation between "Alex" and "Bob" below, generate
{n_q} unique questions that "Bob" can ask "Alex," derived from the
information "Alex" has shared.  Each question should be directly
answerable using the conversation's content.
Output a JSON array where each element is an object with the
following keys:
- "question":  The question for Alex.
- "response":  The corresponding answer derived directly from Alex's
information.
Ensure the output is valid JSON. Only output the JSON array.
[Conversation]
{chat_hist}
[Output]
```

A.5 Further Experimental Results

A.5.1 MSC-E: Accuracy vs Position of Evidence

We present accuracy results on the MSC-E dataset, focusing on how performance varies based on the position of supporting evidence within the dialogue. This analysis demonstrates the model's ability to effectively retrieve and utilize information from different points in extended conversations, highlighting its robustness in scenarios where a memory component is essential for maintaining context.

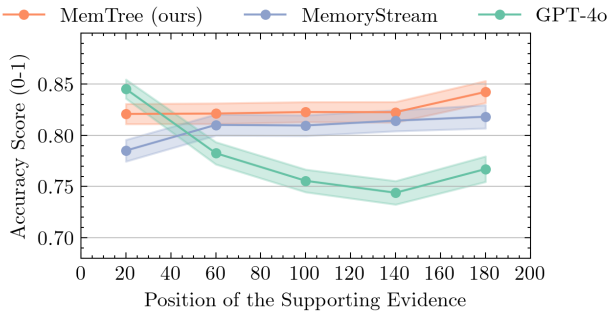

Figure 4: **Accuracy** on MSC-E.

A.5.2 Multihop RAG: Accuracy vs Query Type

We present results across three query types: (1) Inference queries, requiring reasoning from retrieved information; (2) Comparison queries, which involve evaluating and comparing evidence within the retrieved data; and (3) Temporal queries, analyzing time-related information to determine event sequences.

Here, we compare online and offline methods (shaded in gray). In addition to the methods presented in the main paper, two offline approaches—RAG [18] and GraphRAG [8]—are described in the appendix. Note that MemoryStream embeds its memory table using LLM-generated keys that summarize content upon data insertion, while RAG directly embeds based on content. GraphRAG constructs a knowledge graph from LLM-extracted entities and relationships, partitioning the graph into modular communities and summarizing them independently. Responses are then combined using a map-reduce approach to answer global queries. Also, note that offline methods must be rebuilt from scratch to incorporate new information and cannot support real-time memory updates like MemTree.

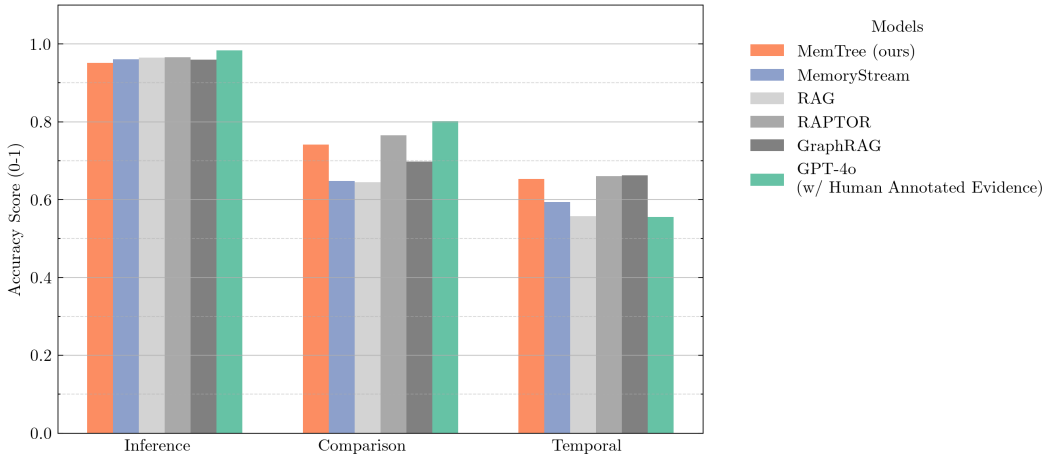

Figure 5: **Accuracy** on MultiHop RAG