Long Context Modeling with Ranked Memory-Augmented Retrieval

Anonymous ACL submission

Abstract

Effective long-term memory management is crucial for language models handling extended contexts. We introduce the Enhanced Ranked Memory Augmented Retrieval (**ERMAR**) framework, which dynamically ranks memory entries based on relevance. Unlike prior models, ERMAR employs a novel relevance scoring mechanism and a pointwise re-ranking model for key-value embeddings, inspired by learning-to-rank techniques in information retrieval. By integrating historical usage patterns and adaptive retrieval, ERMAR achieves state-of-the-art results on standard benchmarks, demonstrating superior scalability and performance in long-context tasks.

1 Introduction

Large Language Models (LLMs) face a fundamental limitation in processing long-context scenarios due to the quadratic complexity of attention mechanisms and increasing memory demands during generation (Vaswani, 2017; Tworkowski et al., 2024). Consider a scenario in an automated customer service system: *A customer reports an issue with their printer, referencing a setup process from a previous conversation that occurred two hours ago. After 50 messages of troubleshooting, the customer mentions that the same error from the beginning has resurfaced. Traditional LLMs, constrained by their context window, would struggle to access the crucial earlier context about the initial setup process, leading to inconsistent or incomplete responses, Figure 1.* It is well known that handling extended contexts remains a significant challenge, particularly in applications requiring document analysis and sustained dialogue interactions.

The recent MemLong (Liu et al., 2024) architecture stores and accesses historical context through basic chunk-level memory operations. The memory bank model is a large, non-trainable store of past context representations. Instead of recomputing representations for all past tokens every time, these representations are pre-computed and stored. Given the current context, MemLong retrieves relevant segments from the memory bank. It uses a dot product similarity search to find the memory entries most related to the current context. This allows the model to focus only on the most pertinent past information, rather than processing the entire history. However, its treatment of all key-value (K-V) pairs with equal weight, regardless of their contextual relevance, often leads to information overload and reduced retrieval precision. This limitation becomes particularly evident in scenarios requiring context management.

We have developed a novel model that addresses the aforementioned limitations by building upon Memlong (Liu et al., 2024), a publicly available baseline on GitHub[1]. Our **Enhanced Ranked Memory Augmented Retrieval** (**ERMAR**) model has a novel relevance scoring mechanism that fundamentally improves context retrieval and utilization for K-V embeddings. Unlike MemLong, ERMAR employs multiplication (Cao et al., 2007) to compute relevance scores, enabling a more nuanced and context-aware assessment of semantic alignment between queries and stored memory. ERMAR also incorporates a re-ranking mechanism that dynamically reorders K-V embeddings based on their relevance scores, ensuring that the most pertinent information is prioritized during retrieval. This re-ranking process, combined with an adaptive retrieval system that integrates historical usage patterns, allows ERMAR to capture subtle contextual relationships better and refine memory prioritization. As shown in Figure 1, ERMAR processes incoming queries and long-context conversations through a novel ranking architecture, employing K-V pairs ranking (K_0-V_0, K_1-V_1,...,K_i-V_i) and cor-

[1]https://github.com/Bui1dMySea/MemLong

responding embeddings to perform semantic search and ranking of relevant historical information.

Our novel ERMAR model introduces three key improvements: (i) A semantic similarity metric to measure contextual alignment between query embeddings and key-value pairs; (ii) A weighted scoring function that considers content similarity and contextual relevance; and (iii) Integration of historical usage patterns to refine relevance assessment.

2 Related Work

The challenge of enabling language models to effectively process extended contexts has driven research across multiple domains. We organize related work into key areas that inform our ERMAR framework.

2.1 Memory-Augmented Neural Networks

Early neural memory architectures like Neural Turing Machines (Graves et al., 2014) and Differentiable Neural Computers (Graves et al., 2016) established the principle of external memory mechanisms beyond parameter storage. Recent memory-augmented architectures have focused on language modeling tasks. RetroMAE (Xiao et al., 2022) demonstrated promising results but struggled with semantic coherence when retrieved passages lacked contextual relevance. Memorizing Transformers (Wu et al., 2022) introduced dedicated memory tokens for cross-sequence information storage, showing improvements on long-range dependency tasks but facing scalability limitations. The recent MemLong architecture (Liu et al., 2024) represents a significant advance by storing historical context through chunk-level memory operations using a non-trainable memory bank. However, its uniform treatment of all key-value pairs regardless of contextual relevance often leads to information overload and reduced retrieval precision, particularly in scenarios requiring nuanced context management.

2.2 Retrieval-Augmented Generation and Long-Context Modeling

Retrieval-augmented generation (RAG)(Lewis et al., 2020) pioneered integrating dense passage retrieval with generative models, while Fusion-in-Decoder (FiD)(Izacard and Grave, 2020) improved efficiency through independent passage processing. REALM (Guu et al., 2020) introduced end-to-end learning of retrieval and generation, and DPR (Karpukhin et al., 2020) established dense passage retrieval standards. However, these methods typically focus on static corpora rather than dynamic context-aware memory management. The quadratic complexity of attention mechanisms has driven research into efficient long-context architectures. Sparse attention mechanisms such as Longformer (Beltagy et al., 2020) and BigBird (Zaheer et al., 2020) reduce computational complexity while maintaining model capabilities through selective attention patterns. Position encoding adaptations like RoPE (Su et al., 2024) and ALiBi (Press et al., 2022) have enhanced models' ability to handle longer sequences. YARN (Peng et al., 2023) further advanced this through dynamic position embeddings, demonstrating reliable generalization up to 128k tokens.

2.3 Learning-to-Rank and Information Retrieval

Our ERMAR framework draws inspiration from learning-to-rank techniques in information retrieval. Traditional ranking approaches like BM25 struggle with semantic similarity, leading to neural ranking models using learned representations. Pointwise ranking approaches (Cao et al., 2007) predict relevance scores for query-document pairs, directly inspiring our relevance scoring mechanism. Dense retrieval methods like DPR (Karpukhin et al., 2020) and ColBERT (Khattab and Zaharia, 2020) demonstrate the effectiveness of learned dense representations for ranking through similarity scores—a paradigm we adapt for memory entry ranking. BERT-based re-ranking models (Nogueira and Cho, 2019) show that sophisticated re-ranking significantly improves retrieval quality. This two-stage retrieve-then-rerank paradigm directly influences our design, where we first retrieve candidate memory entries then apply learned re-ranking.

2.4 Current Limitations and Gaps

Despite significant progress, current approaches face limitations that motivate ERMAR:

Static Memory Management: Most approaches use fixed memory structures that don't adapt to content importance or usage patterns, leading to inefficient memory utilization.

Uniform Memory Treatment: Existing methods treat all memory entries equally, lacking mechanisms to prioritize more relevant information.

Limited Semantic Understanding: Memory systems often rely on simple similarity metrics

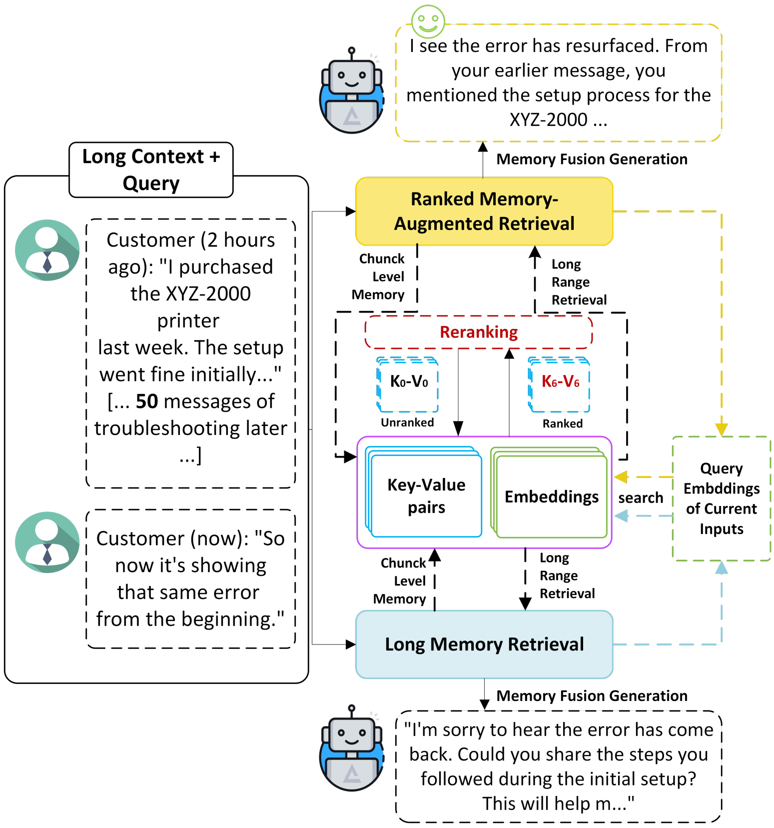

Figure 1: Our novel proposed ERMAR system. Note the difference from the MemLong architecture where we have introduced a novel *Reranking* model.

without considering contextual nuance or temporal relevance.

Scalability Trade-offs: Current approaches face difficult trade-offs between memory capacity, retrieval accuracy, and computational efficiency.

Our ERMAR framework addresses these limitations through dynamic relevance scoring, adaptive memory management, and sophisticated re-ranking mechanisms inspired by information retrieval techniques, providing a more principled approach to long-context memory management.

3 Our Novel ERMAR Model

Figure 2 illustrates our ERMAR framework and Figure 1 presents the contextual ranking mechanism of key, value pairs components that enable effective retrieval. ERMAR maintains consistency through frozen lower layers and selective parameter updating. ERMAR: (i) stores important information from earlier parts of the text; (ii) assigns relevance scores to stored information based on its importance to the current context, and (iii) retrieves only the most relevant historical information when needed. Our relevance scoring is analogous to attention, allowing the model to focus on important parts of the memory. There is also a "loose" point-wise connection because the primary objective is sequence likelihood.

Let $\mathcal{V}$ be a finite vocabulary, and

$$\mathbf{x} = (x_1, \dots, x_n) \in \mathcal{V}^n$$

a token sequence with preceding context $\mathbf{x}_{<i}$. The embedding function

$$\mathcal{E} : \mathcal{V}^* \to \mathbb{R}^{d_{\text{ret}}}$$

maps sequences to retrieval space. We introduce a memory function $\mathcal{M}$ augmented with a relevance scoring mechanism:

$$\mathcal{M} : \underbrace{\mathbb{R}^{d_{\text{model}}}}_{\text{keys}} \times \underbrace{\mathbb{R}^{d_{\text{model}}}}_{\text{values}} \times \underbrace{\mathbb{R}^{d_{\text{ret}}}}_{\text{embeddings}} \to \mathcal{S}$$

Relevance Score: Given a query embedding $\mathbf{q} \in \mathbb{R}^{d_{\text{ret}}}$ and a matrix of key embeddings

$$\mathbf{K} = [k_1, \dots, k_m] \in \mathbb{R}^{m \times d_{\text{ret}}}$$

(where each row $k_i \in \mathbb{R}^{d_{\text{ret}}}$ corresponds to a key embedding), the relevance score is:

$$\alpha(\mathbf{q}, \mathbf{K}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{d_{\text{ret}}}}\right) \tag{1}$$

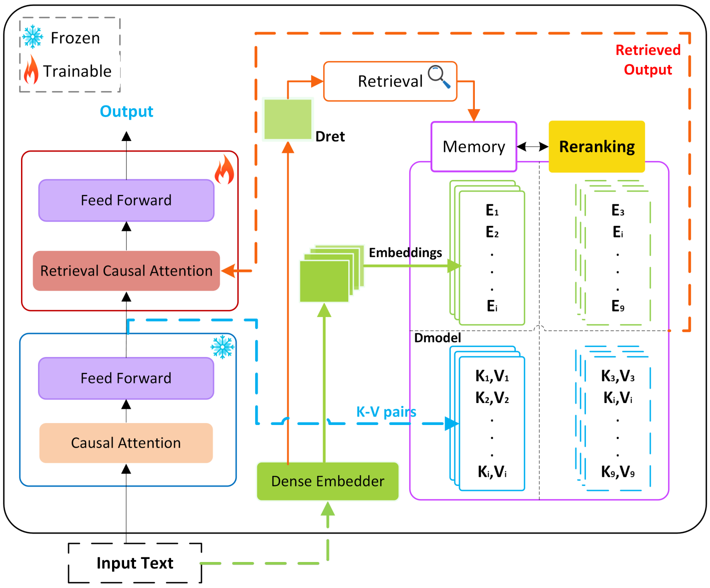

Figure 2: The architecture diagram for ERMAR.

Here, $\sqrt{d_{\text{ret}}}$ normalizes the similarity scores to prevent excessively large values. The relevance score $\alpha(\mathbf{q}, \mathbf{K})$ can be interpreted as a probability distribution over the keys, where each entry α_i represents the relative importance (or attention weight) of the i-th key to the query $\mathbf{q}$. This score is used to rank memory entries based on their importance to the current query.

Ranked Key-Value Pairs: Each embedding $\mathbf{e}$ maintains a ranked set of key-value pairs:

$$\mathcal{R}_{\text{ranked}}(\mathbf{e}) = \{(K_j, V_j, s_j)\}_{j=1}^{m}$$

where $s_j = \alpha(\mathbf{e}, K_j)$ is the relevance score between the embedding $\mathbf{e}$ and the key K_j.

ERMAR objective function is formulated as: Given a sequence $\mathbf{x}$, maximize:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} p_\theta(x_i \mid \mathcal{R}_{\text{RSAR}}(t_i, s), \mathbf{x}_{<i})$$

subject to:

$$\begin{aligned} s_i &= \mathcal{M}(K_{1:i-1}, V_{1:i-1}; \mathcal{E}(t_{1:i-1})) \\ t_i &= \text{text}(c_{\lceil i/\tau \rceil}) \\ \alpha_i &= \alpha(\mathcal{E}(t_i), K_{1:i-1}) \end{aligned}$$

where α_i guides key-value pair selection, and p_θ represents the model's probability distribution.

For new content (K_n, V_n), update the memory state as:

$$s_{i+1} = \begin{cases} \mathcal{M}(K_\text{n}, V_\text{n}; \mathcal{E}(t_\text{n})) & \text{if } |s_i| < \text{capacity}, \\ M_\text{u}(s_i, K_\text{n}, V_\text{n}, \alpha_\text{n}) & \text{otherwise}, \end{cases}$$

where M_u prunes the least relevant entries based on historical scores, specifically by ranking the scores and pruning those with the lowest values relative to the current context.

We now develop the **Relevance Scoring with Adaptive Retrieval (RSAR)**. This approach dynamically ranks memory entries based on their importance to the current query, significantly improving the retrieval process. The relevance score, $\alpha(\mathbf{q}, \mathbf{K})$, as defined in equation 1, is used to rank memory entries.

RSAR enhances the memory module by introducing ranked key-value entries, represented as (K_j, V_j, s_j), where s_j denotes the relevance score for each entry. These scores enable the system to prioritize the most relevant information during retrieval while maintaining computational efficiency. To ensure optimal memory utilization, a pruning strategy is applied to remove less relevant entries. Specifically, entries with scores below a predefined threshold are discarded, preserving only the most critical context.

The enhanced retrieval mechanism is expressed as:

$$R_{\text{RSAR}}(t_q, s) = \text{TopK}\left\{\text{sim}(E(t_q), e) \cdot \max_j s_j \mid e \in s\right\}$$

where $E(t_q)$ represents the encoded query, and the operation identifies the top-K relevant entries based on their scores. This mechanism efficiently retrieves the most relevant information, even for extended contexts.

3.1 Experimental Setup

3.1.1 Datasets

We fine-tuned ERMAR on the *SlimPajama* dataset (Fu et al., 2024), a high-quality, deduplicated corpus designed for long-context tasks. It contains 84.7K training rows, making it a compact yet effective resource for pre-training and fine-tuning. The dataset was preprocessed with a sliding window approach using 512-token strides to ensure comprehensive coverage of long sequences.

Performance evaluation was conducted on three benchmark datasets: *WikiText-103* (Merity et al., 2016) (4,358 test rows), *PG-19* (Rae et al., 2019) (100 test rows), and *Proof-Pile* (Azerbayev et al., 2023) (46.3K test rows). Performance was measured across context lengths from (1k-32k) tokens, using perplexity on the last 2048 tokens (Yen et al., 2024) following standard evaluation protocols.

3.1.2 Base Model Architecture

We fine-tuned OpenLLaMA-3B, a pre-trained LLM with rotational position encoding (Su et al., 2024), using LoRA (Hu et al., 2021) for parameter-efficient fine-tuning. The model architecture consists of $L = 26$ transformer layers, $H = 32$ attention heads, and hidden dimension $d = 3200$ (note: the original $d = 100$ appears to be a typo). The 13th layer serves as the memory layer where historical context is stored, while layers [14, 18, 22, 26] are augmented with retrieval mechanisms to access stored memories.

3.1.3 Model Configuration

We fine-tuned OpenLLaMA-3B using LoRA (Hu et al., 2021) for parameter-efficient training. The model uses layer 13 as the memory layer for storing historical context, while layers [14, 18, 22, 26] are augmented with retrieval mechanisms. ERMAR employs a memory capacity of 32,768 key-value pairs with BGE-M3 embeddings for semantic similarity computation. Complete training hyperparameters and configuration details are provided in Appendix5.

3.1.4 Baseline Models

ERMAR was evaluated against state-of-the-art models across two parameter scales to ensure comprehensive comparison. The 7B models include LLaMA-2-7B (Touvron et al., 2023b) as a standard transformer baseline, LongLoRA-7B-32k (Chen et al., 2023) which employs sparse attention mechanisms for 32k-token contexts, and YARN-128k-7B (Peng et al., 2023) featuring dynamic position embeddings that support up to 128k tokens.

For the 3B parameter scale, we compared against OpenLLaMA-3B (Touvron et al., 2023a) as the base architecture, LongLLaMA-3B (Tworkowski et al., 2024) evaluated in two retrieval configurations (4 and 18 memory entries), MemLong-3B (Liu et al., 2024) as our direct baseline with chunk-level memory operations, and Phi3-128k (Abdin et al., 2024) which demonstrates strong performance across varying context lengths. This diverse benchmark suite encompasses different long-context strategies including sparse attention, position encoding extensions, and memory-augmented architectures, ensuring robust evaluation of ERMAR's retrieval-based approach against complementary methodologies.

3.1.5 Evaluation Metrics

We employ perplexity as the primary metric for language modeling performance, computed on the final 2048 tokens of each sequence to focus on long-range dependency modeling. For in-context learning tasks, we report accuracy on five natural language understanding benchmarks: SST-2, MR, Subj, SST-5, and MPQA, evaluated in both 4-shot and 20-shot settings. Memory efficiency is assessed through peak GPU memory usage and tokens processed per second, while computational overhead is measured via inference latency across different context lengths.

3.2 Results and Discussion

3.2.1 Long-Context Language Modeling

Following the experimental strategy adopted in (Liu et al., 2024), Table 1 presents the mean perplexity scores of our model across different sequence lengths and datasets, demonstrating its effectiveness in long-context modeling. Evaluation was performed on test splits of three datasets: *WikiText-103* (Merity et al., 2016) (4,358 rows), *PG-19* (Rae et al., 2019) (100 rows), and *Proof-Pile* (Azerbayev et al., 2023) (46.3k rows).

Among 7B models, YARN-128k-7B excels in shorter contexts, while LongLoRA-7B-32k scales effectively to 16k-token sequences, though with some performance degradation. This highlights the trade-off between performance and scalability, guiding model selection based on use-case needs.

The 3B models demonstrate ERMAR's significant advantages in long-context tasks. While OpenLLaMA-3B struggles beyond 4k tokens, and

Model	PG19				Proof-pile				Wikitext-103			
	1k	2k	4k	16k	1k	2k	4k	16k	1k	2k	4k	16k
7B Model												
YARN-128k-7b	7.22	7.47	7.17	-	3.03	3.29	2.98	-	5.71	6.11	5.71	-
LongLoRA-7B-32k	9.76	9.71	10.37	7.62	3.68	3.35	3.23	2.60	7.99	7.83	8.39	5.47
LLaMA-2-7B	10.82	10.06	8.92	-	3.24	3.40	2.72	-	10.82	6.49	5.66	-
3B Model												
Phi3-128k	11.31	9.90	**9.66**	-/9.65	4.25	3.11	2.77	-/3.08	7.54	7.22	7.01	-/7.20
OpenLLaMA-3B	11.60	9.77	$> 10^3$	-	**2.96**	**2.70**	$> 10^3$	-	10.57	8.08	$> 10^3$	-
LongLLaMA-3B*	10.59	10.02	$> 10^3$	-	3.55	3.15	$> 10^3$	-	8.88	8.07	$> 10^3$	-
LongLLaMA-3B†	10.59	10.25	9.87	-	3.55	3.22	**2.94**	-	10.69	8.33	7.84	-
MemLong-3B*	10.66	10.09	$> 10^3$	-	3.58	3.18	$> 10^3$	-	8.72	7.93	$> 10^3$	-
w/ 4K MemLong	10.54	9.95	9.89	**9.64**	3.53	3.16	3.15	**2.99**	8.53	7.92	7.87	7.99
w/ 4K **ERMAR**	**10.32**	**9.75**	9.78	9.81	3.24	2.98	3.03	3.18	**8.42**	**7.61**	**7.62**	**7.80**

Table 1: Perplexity comparison of 7B and 3B models across PG19, Proof-pile, and WikiText-103, using a sliding window evaluation. "-" denotes Out of Memory (OOM) errors, and "x/y" indicates results from single/dual GPU setups. Memory-augmented models are tested with varying capacities. All runs use a single GPU.

Phi3-128k shows more consistent performance, ERMAR achieves competitive results across different sequence lengths. At 2k tokens, ERMAR outperforms MemLong on Proof-pile (2.98 vs 3.16) and consistently maintains strong performance across all datasets. ERMAR shows particular strength in the PG19 dataset, achieving the best performance among 3B models at 1k and 2k tokens (10.32 and 9.75 respectively). Most notably, ERMAR demonstrates exceptional stability in long-context scenarios, with minimal degradation from 4k to 16k tokens - only 0.31% increase in perplexity on PG19 (from 9.78 to 9.81), showcasing its superior scalability. On WikiText-103, ERMAR consistently outperforms other 3B models across all tested sequence lengths, further validating the effectiveness of its enhanced memory retrieval mechanism for long-context modelling.

3.2.2 Scalability to Extended Contexts

ERMAR's scalability was evaluated at 32k tokens (Table 2). ERMAR demonstrates consistent performance advantages across multiple datasets: achieving a perplexity of 9.765 versus 9.858 on PG19 (0.90% improvement) and 7.880 versus 7.938 on WikiText-103 (0.06% improvement). Both methods achieve identical performance on Proof-pile with a perplexity of 3.063. This highlights ERMAR's robustness for ultra-long contexts.

Dataset	ERMAR	MemLong	Difference
PG19	9.765	9.858	0.90%
WikiText-103	7.880	7.938	0.06%
Proof-pile	3.063	3.063	0

Table 2: Perplexity at 32k context length, evaluated on NVIDIA L40S GPU.

3.2.3 In-Context Learning Performance

The results in Table 3 show ERMAR's strong performance across five natural language understanding tasks in both 4-shot and 20-shot settings.

In the 4-shot setting, ERMAR achieves state-of-the-art results across all tasks, outperforming OpenLLaMA and other memory-augmented models. It excels even in challenging tasks like SST-5 and MPQA, maintaining high performance with limited examples. Its stability across different memory configurations highlights its robustness in low-resource scenarios.

ERMAR continues to excel in the 20-shot scenario, achieving top results in tasks like MPQA and Subj, and setting a new benchmark for SST-5. While it lags behind MemLong in a few tasks, ERMAR outperforms it overall, showcasing its scalability with increased examples.

ERMAR consistently performs well across varying context lengths, effectively leveraging memory augmentation. Its ability to scale with more examples and handle both short and long-range dependencies makes it a strong candidate for general-purpose language modelling, advancing the state-of-the-art in language understanding tasks.

3.2.4 Memory Efficiency Analysis

Table 4 compares peak and reserved memory usage for ERMAR and MemLong across various context lengths, evaluated on an NVIDIA L40S GPU.

As visualized in Figure 4, ERMAR consistently uses less memory per token, particularly at longer contexts (16k and 32k), demonstrating its efficiency in memory management. The reduced reserved memory (e.g., 16.61 GB vs. 23.77 GB at 16k) underscores ERMAR's optimized dynamic memory management, mitigating the information overload noted in the main text.

Model	In-C ,In-M	SST-2 ACC↑	MR ACC↑	Subj ACC↑	SST-5 ACC↑	MPQA ACC↑	Avg.
OpenLLaMA	4,N/A	90.7	84.0	58.2	41.0	70.5	68.9
w./ Rag	4,4	90.9	90.5	61.6	39.2	63.2	69.1
LongLLaMA	4,4	90.4	83.9	64.3	40.0	64.2	68.6
MemLong	4,4	91.5	84.5	61.5	41.4	70.2	69.8
ERMAR	4,4	**93.6**	**90.8**	**65.3**	**45.8**	**85.2**	**76.14**
LongLLaMA	4,18	91.4	87.1	59.1	41.0	64.5	68.7
MemLong	4,18	91.0	89.6	61.7	43.5	69.4	71.0
ERMAR	4,18	**93.6**	**90.8**	**65.3**	**45.9**	**85.2**	**76.16**
OpenLLaMA	20,N/A	93.6	91.2	55.4	38.2	66.4	69.0
w./ Rag	20,18	92.2	91.3	75.8	39.8	57.6	71.3
LongLLaMA	20,18	94.1	90.8	64.2	41.4	72.1	72.7
MemLong	20,18	93.5	**93.8**	65.8	43.3	70.6	73.4
ERMAR	20,18	**94.7**	91.7	**82.8**	**47**	**86.5**	**80.54**

Table 3: 4-shot and 20-shot ICL accuracy [%] on 5 NLU tasks (SST-2, MR, Subj, SST-5, MPQA). We compare OpenLLaMA, LongLLaMA, MemLong, and ERMAR. **Note:** In-C = In-Context, In-M = In-Memory.

Context Length	Model	Peak Mem (GB)	Reserved Mem (GB)	Mem/Token (MB)
1024	ERMAR	7.97	8.16	7.97
	MemLong	8.08	8.49	8.08
2048	ERMAR	8.45	8.71	4.22
	MemLong	8.67	9.38	4.33
4096	ERMAR	9.42	9.87	2.35
	MemLong	9.72	10.58	2.43
16384	ERMAR	15.20	16.61	0.95
	MemLong	15.60	23.77	0.97
32768	ERMAR	22.87	25.56	0.71
	MemLong	23.27	26.05	0.72

Table 4: Memory efficiency comparison of ERMAR and MemLong across context lengths on Wikitext data. Mem/Token is calculated as Peak Mem divided by context length.

3.2.5 Latency and Throughput Analysis

Table 5 provides latency and throughput comparisons for ERMAR and MemLong on the PG-19 dataset, evaluated on an NVIDIA L40S (44.4GB) GPU.

Context	Model	Latency	Latency /Token (ms)	Throughput (tokens/ sec)
1024	ERMAR	207.97 ± 53.01	0.203	5135
	MemLong	190.68 ± 154.55	0.186	6803
2048	ERMAR	423.79 ± 52.56	0.206	4896
	MemLong	323.49 ± 204.89	0.157	7446
4096	ERMAR	1358.36 ± 55.58	0.331	3020
	MemLong	1184.16 ± 170.56	0.289	3511
16384	ERMAR	6862.98 ± 45.65	0.418	2387
	MemLong	6496.00 ± 213.56	0.396	2524
32768	ERMAR	13679.03 ± 58.76	0.417	2395
	MemLong	13449.90 ± 56.22	0.410	2436

Table 5: Latency and throughput comparison on PG-19 dataset. Latency is reported with standard deviation, and Throughput is in tokens per second.

ERMAR exhibits higher latency (9–35% at 1k–4k, narrowing to 1.7–5.5% at 16k–32k) but maintains competitive throughput (2.4k tokens/sec at long contexts). Notably, ERMAR's lower latency variance (±45–59ms vs. MemLong's ±154–214ms) indicates greater stability, aligning with its robustness in long-context tasks.

Single-Model Performance Scaling: To complement the comparative analysis, we evaluated ERMAR's standalone performance characteristics across different sequence lengths on WikiText-103 with 16K memory capacity.

Seq Length	Perplexity	Memory /Token (GB)	Throughput (tokens /sec)	Latency /Token (ms)
1K	8.42	7.13	3125	0.32
2K	7.61	3.81	2904	0.35
4K	7.62	2.14	2109	0.47
8K	7.76	1.31	1836	0.54
16K	7.80	0.90	1727	0.58

Table 6: ERMAR's standalone performance scaling on WikiText-103, showing memory efficiency and throughput characteristics across sequence lengths.

These results demonstrate ERMAR's excellent scalability characteristics, with an 8-fold improvement in memory efficiency (7.13 → 0.90 MB/token) while maintaining stable perplexity performance. The throughput values on WikiText-103 are higher than the comparative PG-19 results, likely due to dataset-specific processing characteristics and the different evaluation methodologies used.

3.2.6 Ablation Studies

Ablation Study on Embedders: Following the comprehensive evaluation of BGE and LLM embedders, we expand our analysis to understand the nuanced impact of embedding architectures on ERMAR's performance across different tasks and configurations.

We conducted extensive ablation studies comparing BGE and LLM embedders across SST-2, Subj, SST-5, MPQA, and MR tasks under various configurations, examining Flash vs. Eager attention, different context lengths (0 and 2048), and varying in-context/in-memory demonstration settings. The results are shown in Tables 7 and 8.

The ablation study reveals that both BGE and LLM embedders perform similarly, with the BGE embedder slightly outperforming LLM in the 20-shot, 18 in-memory setting (e.g., 77.80 vs. 71.70 for Eager attention at 0 context length). The performance difference is most pronounced in the Subj task, where BGE achieves up to 82.85% accuracy compared to LLM's 50.35% in some configurations. This suggests that BGE may better capture semantic nuances in certain tasks, though both embedders maintain comparable performance overall. Further investigation into embedder-specific optimizations could enhance ERMAR's performance,

Flash Attention	Context Length	In-Context	In-Memory	SST-2	Subj	SST-5	MPQA	MR	Average
Flash	0	4	4	88.07	57.30	41.42	80.04	83.84	70.13
		4	18	88.07	57.30	41.42	80.33	83.86	70.20
		20	18	93.92	50.35	47.05	73.63	91.50	71.69
Eager	0	4	4	88.19	51.55	41.42	79.92	83.89	69.59
		4	18	88.19	51.55	41.51	80.29	83.92	69.89
		20	18	94.04	82.85	46.96	73.63	91.54	77.80
Flash	2048	4	4	88.07	51.55	41.42	80.04	83.84	68.98
		4	18	88.07	51.25	41.42	80.33	83.86	68.99
		20	18	93.92	82.75	47.05	73.63	91.50	77.77
Eager	2048	4	4	88.19	51.55	41.51	79.86	83.89	68.96
		4	18	88.19	51.55	41.51	79.60	83.92	68.95
		20	18	94.04	82.85	46.96	73.63	91.34	77.76

Table 7: Performance comparison of BGE embedder across configurations on SST-2, Subj, SST-5, MPQA, and MR tasks. Accuracy is reported in percentage.

Flash Attention	Context Length	In-Context	In-Memory	SST-2	Subj	SST-5	MPQA	MR	Average
Flash	0	4	4	88.07	57.30	41.42	80.04	83.84	70.13
		4	18	88.07	57.30	41.42	80.33	83.86	70.20
		20	18	93.92	50.35	47.05	73.63	91.54	71.70
Eager	0	4	4	88.19	57.30	41.51	79.86	83.89	70.15
		4	18	88.19	57.30	41.51	79.60	83.92	70.10
		20	18	94.04	50.35	46.96	73.63	91.54	71.70
Flash	2048	4	4	88.07	51.55	41.42	80.04	83.84	68.98
		4	18	88.07	51.55	41.42	80.33	83.86	69.05
		20	18	93.92	82.85	47.05	73.63	91.50	77.79
Eager	2048	4	4	88.19	51.25	41.51	79.92	83.89	68.95
		4	18	88.19	51.25	41.51	80.29	83.92	69.03
		20	18	94.04	82.75	46.96	73.63	91.34	77.74

Table 8: Performance comparison of LLM embedder across configurations on SST-2, Subj, SST-5, MPQA, and MR tasks. Accuracy is reported in percentage.

particularly for tasks requiring fine-grained contextual understanding.

Ablation Study on Relevance and Reranking:

We conducted detailed ablation studies on the core Relevance+Re-ranking mechanism, evaluating its impact on perplexity across varying context lengths for WikiText-103 and PG19 datasets. The results are summarized in Table 9.

Dataset	Context Length	Without Relevance+Reranker	With Relevance+Reranker
		Perplexity	Perplexity
Wiki	1K	8.841	7.919
	2K	7.984	7.410
	4K	7.438	7.437
	16K	7.267	7.082
	32K	7.938	8.008
PG19	1K	11.451	10.322
	2K	10.412	9.746
	4K	9.932	9.780
	16K	9.910	9.809
	32K	9.858	9.765

Table 9: Ablation study showing the impact of the Relevance+Reranking mechanism on perplexity for the WikiText-103 (Wiki) and PG19 datasets across different context lengths.

Relevance Mechanism Effectiveness: The relevance scoring mechanism shows strongest benefits at shorter context lengths: 10.4% improvement at 1K tokens and 7.2% at 2K tokens for WikiText-103. Benefits diminish at 4K tokens and become slightly negative at 32K tokens, suggesting ranking overhead outweighs advantages when memory capacity is sufficient. PG19 shows more consistent improvements (1.0%-9.9%) across all lengths, indicating narrative text benefits more from semantic ranking as story elements can be referenced non-sequentially.

4 Conclusion

We presented a novel ERMAR framework that enhances long-context modelling through relevance scoring and adaptive memory retrieval. ERMAR outperforms baseline models, including OpenLLaMA, LongLLaMA, and MemLong, achieving superior perplexity in long-context language modeling task and superior accuracy in in-context learning. Future work will focus on optimizing ERMAR for specialized datasets and expanding its applicability to complex reasoning tasks.

5 Limitations

While ERMAR improves retrieval efficiency and context retention, it has limitations. Its reliance on ranked memory structures increases computational overhead compared to standard LLMs, particularly for large-scale retrieval as discussed in A.5 and A.6 in appendix sections. Additionally, performance variations across different task domains indicate a need for further tuning. The framework's effectiveness in real-world, noisy environments also requires further validation.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Zhangir Azerbayev, Edward Ayers, and Bartosz Piotrowski. 2023. Proofpile: A pre-training dataset of mathematical texts.

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

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128k context. *arXiv preprint arXiv:2402.10171*.

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

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

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

Weijie Liu, Zecheng Tang, Juntao Li, Kehai Chen, and Min Zhang. 2024. Memlong: Memory-augmented retrieval for long text modeling. *arXiv preprint arXiv:2408.16967*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. *Preprint*, arXiv:2108.12409.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2024. Focused transformer: Contrastive training for context scaling. *Advances in Neural Information Processing Systems*, 36.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. *arXiv preprint arXiv:2203.08913*.

Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. Retromae: Pre-training retrieval-oriented language models via masked auto-encoder. *arXiv preprint arXiv:2205.12035*.

Howard Yen, Tianyu Gao, and Danqi Chen. 2024. Long-context language modeling with parallel context encoding. *arXiv preprint arXiv:2402.16617*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Appendix

This appendix supplements the main text with detailed implementation and evaluation specifics for the Enhanced Ranked Memory Augmented Retrieval (ERMAR) framework. It includes comprehensive descriptions of dataset preprocessing, training configurations, hyperparameters, and key terminology, along with extended analyses of context-length performance and memory efficiency. The provided details ensure reproducibility and offer deeper insights into ERMAR's architectural and operational nuances.

A.1 Dataset Preprocessing

The *SlimPajama* dataset, used for fine-tuning ERMAR, underwent several preprocessing steps to ensure compatibility with long-context tasks. The dataset was tokenized using the OpenLLaMA tokenizer, with a maximum sequence length of 32768 tokens. Duplicate sequences were removed using a hash-based deduplication algorithm, reducing the dataset to 84.7K unique training rows. To handle variable context lengths, we applied a sliding window approach with a stride of 512 tokens, ensuring that the model could process contexts ranging from 1024 to 32768 tokens. Special tokens were added to denote document boundaries, and padding was applied to align sequences to the nearest multiple of 128 tokens for efficient batch processing.

Training Configuration

ERMAR was trained using a two-stage fine-tuning approach with the following configuration:

Model Parameters:

- Base model: OpenLLaMA-3B-v2 with LoRA adaptations
- Memory layer: Layer 13 for historical context storage
- Retrieval attention layers: [14, 18, 22, 26]
- Memory capacity: 32,768 key-value pairs
- Memory group size: 128 tokens per memory group
- Retrieval group size: 8 (TopK retrieval)
- Gate mechanism: Disabled (use_gate=False)

Training Hyperparameters:

- Learning rate: 5×10^{-5} with 1,000 warmup steps
- Weight decay: 1×10^{-4}
- Batch size: 1 per device (with gradient accumulation)
- Sequence length: 1,024 tokens
- Last context length: 1,024 tokens
- Training epochs: 1 epoch on SlimPajama 0.5B subset
- Training mode: LoRA-freeze (partial parameter updates)

LoRA Configuration:

- Target modules: q_proj, k_proj, v_proj, o_proj
- Trainable parameters: Layer normalization and embeddings
- Frozen layers: Layers 0-13 (up to memory layer)
- Position encoding: Zero position type for extended contexts

Embedder Setup:

- Embedder: BAAI/bge-m3 (BGE embedder)
- Embedding dimension: 1,024
- GPU-based similarity search for efficient retrieval

Hardware and Infrastructure:

- Primary training: Single NVIDIA 3090 24GB GPU
- Extended context (32k): NVIDIA L40S 44.4GB GPU
- Distributed training: ZeRO-2 optimization with Accelerate
- Memory optimization: Sequential batching and continual fine-tuning

A.3 Glossary of Terms

To aid understanding, we provide definitions for key terms used in the ERMAR framework:

- **Relevance Score** (α): A normalized score computed via softmax over the dot product of query and key embeddings, representing the contextual importance of a memory entry (see Equation 1).
- **Key-Value Pair**: A tuple (K_j, V_j) storing contextual information, where K_j is the key embedding and V_j is the corresponding value embedding in the memory bank.
- **RSAR (Relevance Scoring with Adaptive Retrieval)**: The mechanism that dynamically ranks key-value pairs based on their relevance to the query, incorporating historical usage patterns.
- **Memory Bank**: A non-trainable storage of pre-computed key-value embeddings, used to retain historical context without recomputation.
- **TopK Retrieval**: The process of selecting the top-K most relevant memory entries based on their relevance scores for use in the current context.

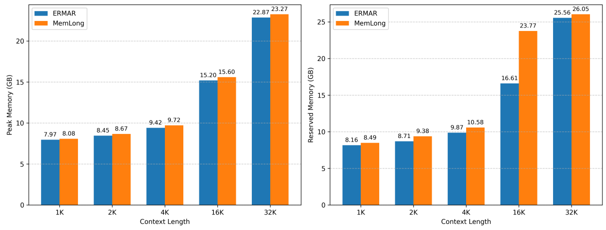

Figure 4: Memory usage comparison between ERMAR and MemLong across context lengths (1K-32K tokens), showing (left) peak memory and (right) reserved memory. ERMAR demonstrates 7-30% lower reserved memory requirements at longer contexts.

A.4 Fine-Grained Context Length Analysis

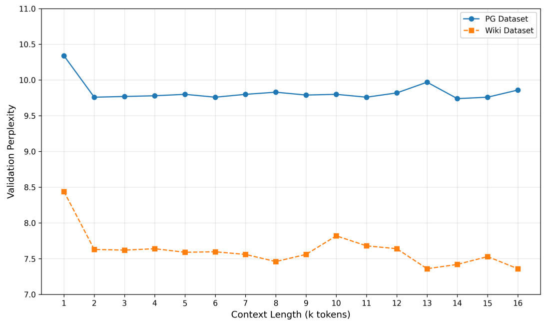

Figure 3: ERMAR perplexity performance across fine-grained context lengths for PG-19 and WikiText-103 datasets. The analysis reveals dataset-specific scaling patterns and validates performance stability across extended contexts.

Figure 3 presents detailed perplexity measurements across incremental context lengths from 1000 to 16000 tokens, providing fine-grained insights into ERMAR's scaling behavior. The fine-grained analysis reveals that WikiText-103 achieves optimal performance around 13K-16K tokens (perplexity 7.36), while PG-19 maintains consistent performance (9.74-9.97) across all context lengths. This validates ERMAR's robustness and suggests that factual content benefits more from extended context than narrative text.

A.5 Memory Efficiency

Figure 4 validates ERMAR's memory optimization advantages, particularly for sequences beyond 8K tokens where it reduces reserved memory by 16.61GB vs. 23.77GB (16K context) compared to MemLong. This demonstrates more effective dynamic memory allocation during extended context processing.

A.5 Relevance+Reranker Visual Analysis

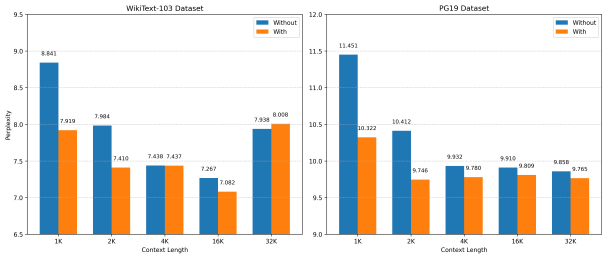

Figure 5: Visualization of Table 9, showing the relative impact of the Relevance+Reranker mechanism across context lengths. Color coding highlights: blue bars represent baseline performance, while orange bars show improvements with our full mechanism(ERMAR).

Figure 5 provides complementary visual evidence for the patterns discussed in Section 3.2.6:

- The **stronger improvements at shorter contexts** (left-side bars) are visually apparent through larger orange/blue differentials
- **Dataset differences** in mechanism effectiveness become immediately visible through side-by-side comparison
- The **32K edge case** (WikiText) where the mechanism underperforms stands out graphically