Long Context Modeling with Ranked Memory-Augmented Retrieval

Ghadir Alselwi[1], Hao Xue[1,2,3], Shoaib Jameel[4], Basem Suleiman[1], Flora D. Salim[1], Imran Razzak[5,1]
[1]University of New South Wales, Sydney, NSW, Australia
[2]Hong Kong University of Science and Technology (Guangzhou)
[3]The Hong Kong University of Science and Technology
[4]University of Southampton, Southampton, UK
[5]Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE
{g.alselwi,b.suleiman,flora.salim}@unsw.edu.au
haoxue@hkust-gz.edu.cn, M.S.Jameel@southampton.ac.uk, imran.razzak@mbzuai.ac.ae

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. Code will be released at https://github.com/cruiseresearchgroup/ERMAR.

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

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

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3576–3590
July 2-7, 2026 ©2026 Association for Computational Linguistics

K-V pairs ranking (K_0-V_0, K_1-V_1,...,K_i-V_i) and corresponding 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 architectures like Neural Turing Machines (Graves et al., 2014) and Differentiable Neural Computers (Graves et al., 2016) introduced external memory beyond model parameters. Recent models for long-context language modeling include RetroMAE (Xiao et al., 2022), which struggled with semantic coherence, and Memorizing Transformers (Wu et al., 2022), which improved cross-sequence information storage but faced scalability limits. MemLong (Liu et al., 2024) stores historical context via chunk-level memory operations, but its uniform treatment of key-value pairs can cause information overload and reduced retrieval precision.

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 and 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 do not 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 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

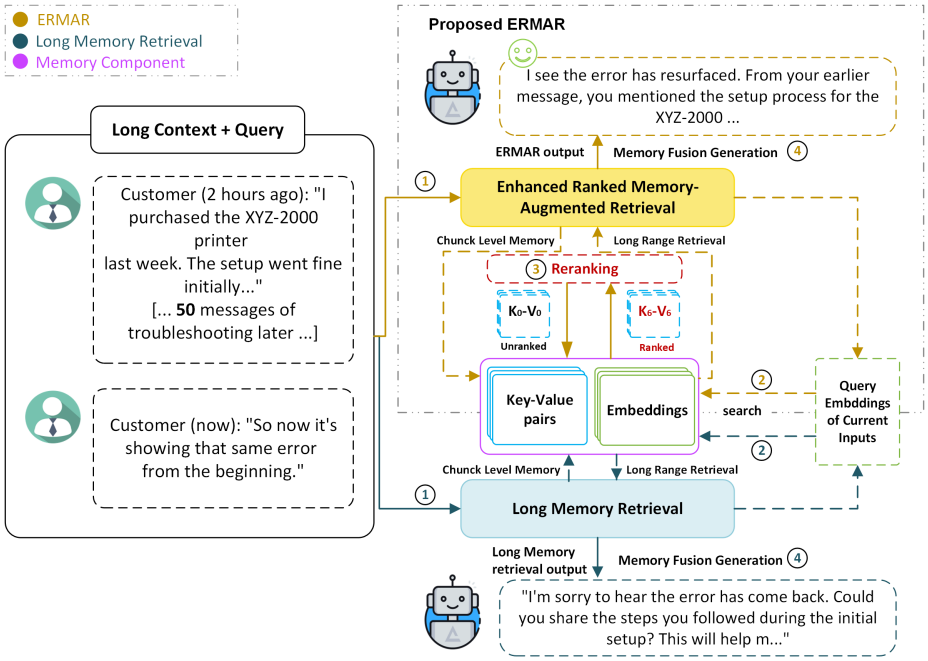

Figure 1: Motivating example of the ERMAR retrieval workflow. *Left*: A long customer support conversation where the current query references information from much earlier context. *Right*: The four-stage workflow — ① Long Memory Retrieval, ② Search, ③ Reranking, ④ Memory Fusion Generation. The ERMAR output (top) correctly references earlier context, whereas the output without reranking (bottom) produces a generic response.

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 1 presents a concrete motivating example of how ERMAR's contextual ranking mechanism operates in practice, while Figure 2 illustrates the overall architecture of the proposed framework. As shown in Figure 1, ERMAR processes a long-context query through four sequential stages. First, the **Long Memory Retrieval** module encodes the full input history into chunk-level key–value (KV) pairs stored in a memory bank, alongside embeddings that capture semantic representations of each chunk. Second, when a new query arrives, its embedding is used to **search** the memory bank for candidate KV pairs via similarity matching. Third, the retrieved candidates pass through the **Reranking** module — the core novelty over MemLong — which assigns relevance scores to each KV pair and re-orders them by contextual importance, elevating high-relevance entries (K_6-V_6, ranked) above lower-relevance ones (K_0-V_0, unranked). Finally, only the top-ranked KV pairs are passed to **Memory Fusion Generation**, which conditions the model's output on the most relevant historical context.

This design directly addresses the information dilution problem in long contexts: without reranking (bottom of Figure 1), the model produces a generic response that fails to reference the earlier printer setup context; with reranking (ERMAR output, top), the model correctly retrieves and incorporates the relevant earlier information.

As shown in Figure 2, the ERMAR architecture consists of three interacting components: a **frozen lower transformer block**, a **trainable upper block** with retrieval-augmented attention, and a **memory retrieval module** with reranking. Input text is processed by the frozen lower block (Causal Attention and Feed Forward layers), whose hidden states form the key–value pairs $\{(K_j, V_j)\} \in \mathbb{R}^{d_{\text{model}}}$ stored in the Memory Bank. In parallel, the **Dense Embedder** independently encodes the same input into chunk-level retrieval embeddings $E_1, \ldots, E_i \in \mathbb{R}^{d_{\text{ret}}}$, which serve as the semantic index for similarity-based retrieval. The **Reranking** module then scores all stored embeddings and KV pairs using Equation 2, re-ordering them by relevance to the current query and isolating the top-ranked subset (right stacks in Figure 2). The **Retrieval** module performs top-K selection over

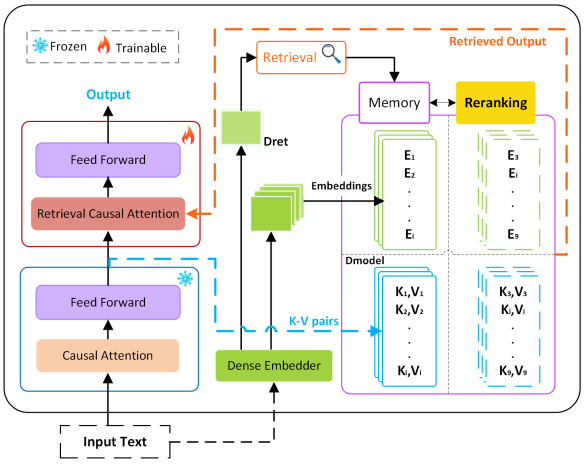

Figure 2: Architecture of the ERMAR model. The frozen lower block (blue dashed, snowflake) produces hidden states stored as key–value pairs (d_{model}) in the Memory Bank, while the **Dense Embedder** encodes the input into chunk-level retrieval embeddings (d_{ret}). The **Reranking** module scores and ranks all stored entries, and the **Retrieval** module injects the top-K ranked pairs into the trainable upper block via **Retrieval Causal Attention** (orange arrow) to condition the final output. Frozen and trainable components are marked by snowflake and flame icons respectively.

this ranked subset and injects the selected KV pairs into the trainable upper block via **Retrieval Causal Attention** (orange arrow), producing a context-enriched representation from which the final output is generated.

ERMAR maintains consistency through frozen lower layers and selective parameter updates. Specifically, 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. The relevance scoring mechanism is analogous to the attention operation, enabling the model to focus on salient parts of the memory. There is also a "loose" pointwise connection: while relevance-based reranking acts as an auxiliary mechanism that enhances memory retrieval quality, the primary optimization target remains next-token prediction likelihood.

Let $\mathcal{V}$ be a finite vocabulary, and $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{V}^n$ be a token sequence with a preceding context $\mathbf{x}_{<i}$. The embedding function $\mathcal{E} : \mathcal{V}^* \to \mathbb{R}^{d_{\text{ret}}}$ maps sequences to a retrieval space. We introduce a memory function $\mathcal{M}$ augmented with a relevance scoring mechanism,

$$\mathcal{M} : \underbrace{\mathbb{R}^{d_{\text{model}}}}_{\text{keys}} \times \underbrace{\mathbb{R}^{d_{\text{model}}}}_{\text{values}} \times \underbrace{\mathbb{R}^{d_{\text{ret}}}}_{\text{embeddings}} \to \mathcal{S}, \tag{1}$$

where d_{model} denotes the transformer hidden dimension, d_{ret} the retrieval embedding dimension, and $\mathcal{S}$ the space of scored memory items (see Appendix A.8 for embedding dimensionality analysis).

Relevance Score. Given a query embedding $\mathbf{q} \in \mathbb{R}^{d_{\text{ret}}}$ and a matrix of key embeddings $\mathbf{K} = [k_1, \ldots, k_m] \in \mathbb{R}^{m \times d_{\text{ret}}}$, where each row $k_i \in \mathbb{R}^{d_{\text{ret}}}$ corresponds to a key embedding, the relevance score is defined as:

$$\alpha(\mathbf{q}, \mathbf{K}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{d_{\text{ret}}}}\right), \tag{2}$$

where $\sqrt{d_{\text{ret}}}$ normalizes the similarity scores to prevent excessively large values.

The resulting vector $\alpha(\mathbf{q}, \mathbf{K})$ represents a probability distribution over the keys, where each entry α_i denotes the relative importance (attention weight) of the i-th key with respect to the query $\mathbf{q}$. The final context or weighted output is obtained as $\mathbf{o} = \sum_{i=1}^{m} \alpha_i V_i$, where V_i denotes the corresponding value embedding associated with key k_i.

We now introduce the **Relevance Scoring with Adaptive Retrieval (RSAR)** mechanism, which dynamically ranks memory entries according to their importance to the current query, thereby enhancing the retrieval process. The relevance score $\alpha(\mathbf{q}, \mathbf{K})$, as defined in Equation 2, is employed to rank the memory entries.

RSAR augments 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 allow the system to prioritize the most informative content during retrieval while maintaining computational efficiency. To optimize memory usage, 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 contextual information.

The enhanced retrieval mechanism is defined as:

$$R_{\text{RSAR}}(t_q, s) = \text{TopK}\left\{\text{sim}(E(t_q), e) \cdot \max_j s_j \mid e \in s\right\}, \tag{3}$$

where t_q denotes the query token, $E(t_q)$ represents the encoded query, and the operation identifies the top-K relevant entries based on their scores; this mechanism enables efficient retrieval of the most

relevant information, even in extended contexts (see Appendix A.6 for the impact of different top-K values on performance).

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

$$\mathcal{R}_{\text{ranked}}(\mathbf{e}) = \{(K_j, V_j, s_j)\}_{j=1}^{m}, \tag{4}$$

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

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

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} p_\theta(x_i \mid \mathcal{R}_{\text{RSAR}}(t_i, s), \mathbf{x}_{<i}), \tag{5}$$

subject to

$$s_i = \mathcal{M}(K_{1:i-1}, V_{1:i-1}; \mathcal{E}(t_{1:i-1})), \tag{6}$$
$$t_i = \text{text}(c_{\lceil i/\tau \rceil}), \tag{7}$$
$$\alpha_i = \alpha(\mathcal{E}(t_i), K_{1:i-1}), \tag{8}$$

where c denotes chunk indices, τ represents the chunk size, α_i guides key–value pair selection, and p_θ denotes the model's probability distribution.

For new content $(K_{\text{n}}, V_{\text{n}})$, the memory state is updated as:

$$s_{i+1} = \begin{cases} \mathcal{M}(K_{\text{n}}, V_{\text{n}}; \mathcal{E}(t_{\text{n}})) & \text{if } |s_i| < \text{capacity}, \\ M_{\text{u}}(s_i, K_{\text{n}}, V_{\text{n}}, \alpha_{\text{n}}) & \text{otherwise}, \end{cases} \tag{9}$$

where capacity is the maximum allowed number of memory entries, M_{u} prunes the least relevant entries based on historical scores by ranking and removing those with the lowest relevance relative to the current context, and subscript n denotes the new key, value, and relevance score corresponding to the latest input.

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 Model Architecture and Configuration

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 adaptation. The model consists of $L = 26$ transformer layers, $H = 32$ attention heads, and a hidden dimension of $d = 100$. 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. ERMAR employs a memory capacity of 32,768 key–value pairs and utilizes BGE-M3 embeddings for semantic similarity computation. Complete training hyperparameters and configuration details are provided in Appendix A.2.

3.1.3 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.

3.1.4 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 to-

kens 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 reports the mean perplexity scores of ERMAR across various sequence lengths and datasets. Evaluation was conducted on the test splits of three standard benchmarks: *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 the 7B models, YARN-128k-7B achieves the lowest perplexity on shorter contexts, whereas LongLoRA-7B-32k scales effectively up to 16k-token sequences, albeit with mild performance degradation. This result highlights the trade-off between scalability and modeling precision commonly observed in long-context transformers.

For the 3B parameter models, ERMAR demonstrates substantial gains in long-context scenarios. While OpenLLaMA-3B exhibits sharp degradation beyond 4k tokens and Phi3-128k maintains moderate stability, ERMAR achieves consistently competitive performance across all sequence lengths. At 2k tokens, ERMAR surpasses MemLong on *Proof-Pile* (2.98 vs. 3.16) and sustains strong results across the remaining datasets. Its advantage is particularly evident on *PG-19*, where ERMAR achieves the best performance among 3B models at 1k and 2k tokens (10.32 and 9.75, respectively). Notably, ERMAR shows exceptional stability when scaling to longer contexts, with only a 0.31% increase in perplexity from 4k to 16k tokens on *PG-19* (9.78 $\rightarrow$ 9.81), indicating superior scalability. On *WikiText-103*, ERMAR consistently outperforms other 3B models at all evaluated context lengths, further validating the effectiveness of its relevance-based memory retrieval mechanism for long-context modeling.

Overall comparison against MemLong. Across all 12 configurations in Table 1 (3 datasets $\times$ 4 context lengths), ERMAR outperforms MemLong in **10 out of 12 settings (83.3%)**. The two exceptions occur at 16k tokens on *PG-19* (9.81 vs. 9.64, $+1.7\%$) and *Proof-Pile* (3.18 vs. 2.99, $+6.4\%$), where the overhead of the relevance scoring and reranking mechanisms marginally outweighs the retrieval benefit at very long contexts. Crucially, these are edge cases at the boundary of the training distribution (the model was fine-tuned up to 32k tokens): ERMAR maintains consistent improvements at practical context lengths (1K–4K), where memory pressure is highest, and fully recovers at 32k tokens (Table 2), achieving equal or better perplexity across all three datasets. The 16k degradation therefore reflects an expected efficiency–benefit trade-off rather than a systematic failure of the proposed enhancements.

These findings confirm that ERMAR's ranked memory retrieval effectively mitigates information dilution across extended sequences, improving both accuracy and stability over prior memory-augmented models.

3.2.2 Scalability to Extended Contexts

To further assess scalability, we evaluated ERMAR at a 32k-token context length (Table 2). ERMAR maintains consistent performance advantages across all datasets, achieving a perplexity of 9.765 compared to 9.858 on *PG19* (a 0.90% improvement) and 7.880 versus 7.938 on *WikiText-103* (a 0.06% improvement). Both models yield identical results on *Proof-Pile* with a perplexity of 3.063. These results demonstrate ERMAR's robustness and scalability in ultra-long context settings, reflecting the effectiveness of its ranked memory retrieval in preserving contextual coherence over extended sequences.

3.2.3 In-Context Learning Performance

Table 3 summarizes ERMAR's performance across five natural language understanding tasks under 4-shot and 20-shot settings.

In the 4-shot configuration, ERMAR achieves superior accuracy across all tasks, outperforming OpenLLaMA and other memory-augmented baselines. It performs notably well on challenging benchmarks such as *SST-5* and *MPQA*, demonstrating strong generalization even with limited examples. The consistent results across different memory configurations further indicate its robustness in low-resource scenarios.

In the 20-shot setting, ERMAR continues to deliver leading results, achieving top accuracy on *MPQA* and *Subj*, and establishing a new benchmark on *SST-5*. Although MemLong marginally surpasses ERMAR in isolated tasks, ERMAR maintains the highest overall average performance, reflecting its scalability as the number of in-context examples increases.

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.

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.

Model	InC ,InM	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:** InC = In-Context, InM = In-Memory.

Overall, ERMAR consistently performs well across varying context lengths and task complexities, effectively leveraging ranked memory retrieval to enhance representation quality. Its ability to scale with more examples and preserve performance stability across tasks underscores its potential as a general-purpose framework for language understanding and long-context modeling.

3.2.4 Memory Efficiency Analysis

Table 4 compares the peak and reserved memory usage of ERMAR and MemLong across different context lengths.

As shown in Table 4 and Figure 3, ERMAR exhibits clear memory optimization advantages, particularly beyond 8K tokens. At a 16K context, ERMAR reduces reserved memory from 23.77 GB to 16.61 GB, corresponding to a 30% improvement over MemLong. It also maintains lower memory consumption per token across all evaluated lengths, demonstrating superior scalability and efficiency in memory management. This improvement stems from ERMAR's ranked retrieval and adaptive memory allocation, which reduce redundancy in long-context storage.

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-103*. Mem/Token is computed as Peak Memory divided by context length.

Memory Pruning Strategy. ERMAR further improves efficiency through a dynamic memory pruning mechanism. The algorithm balances three criteria: retaining the most recent 10% of memory, prioritizing the middle 80% based on retrieval frequency, and discarding the oldest 10% as potentially outdated. This hybrid strategy allows ERMAR to adaptively manage memory content according to actual usage patterns, ensuring that valuable information is preserved while minimizing unnecessary memory overhead. Such adaptive pruning complements the ranked retrieval process, enhancing memory efficiency especially at longer context lengths. A systematic validation of these ratios across 11 pruning strategies and five context lengths is provided in Appendix A.7.

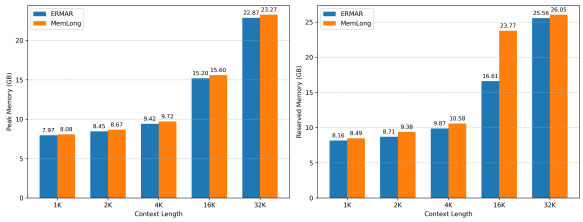

Figure 3: Memory usage comparison between ERMAR and MemLong across context lengths (1K–32K tokens), showing (left) peak memory and (right) reserved memory. ERMAR demonstrates 7–30% lower reserved memory requirements at longer contexts. Color coding: blue bars represent MemLong performance, while orange bars show ERMAR improvements.

3.2.5 Latency and Throughput Analysis

Table 5 compares the latency and throughput of ERMAR and MemLong on the *PG-19* dataset.

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 the *PG-19* dataset. Latency is reported with standard deviation, and throughput is measured in tokens per second.

ERMAR exhibits slightly higher latency (9–35% relative increase at 1K–4K tokens, narrowing to 1.7–5.5% at 16K–32K) while maintaining competitive throughput of approximately 2.4K tokens/s at longer contexts. Importantly, ERMAR demonstrates much lower latency variance (±45–59 ms) compared to MemLong (±154–214 ms), reflecting greater runtime stability in long-context settings. This reduced variance aligns with ERMAR's adaptive ranked retrieval strategy, which streamlines key–value access and minimizes redundant memory operations.

Single-Model Performance Scaling. ERMAR's standalone performance across sequence lengths on the *WikiText-103* dataset (16K memory capacity) is presented in Table 6.

These results highlight ERMAR's scalability, achieving an eightfold improvement in per-token memory efficiency (7.13 MB → 0.90 MB) while maintaining stable perplexity across increasing sequence lengths. The slightly higher throughput observed on *WikiText-103* compared to *PG-19* likely stems from dataset-specific characteristics and differences in evaluation methodology.

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 across sequence lengths.

3.2.6 Ablation Studies

Ablation on Embedders. To investigate the influence of embedding architectures on ERMAR's performance, we compare BGE and LLM embedders across five NLU tasks (*SST-2*, *Subj*, *SST-5*, *MPQA*, and *MR*) under various configurations. Experiments examine the effects of Flash versus Eager attention modes, different context lengths (0 and 2048), and varying In-C/In-M demonstration settings. Results are presented in Tables 7 and 8.

Att	CLen	InC	InM	SST2	Subj	SST5	MPQA	MR	Avg
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 the BGE embedder across configurations on five NLU tasks. Accuracy is reported in percentage. Abbreviations: Att = Attention, CLen = Context length, Avg = Average.

Att	CLen	InC	InM	SST2	Subj	SST5	MPQA	MR	Avg
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 the LLM embedder across configurations on five NLU tasks. Accuracy is reported in percentage.

The results indicate that both BGE and LLM embedders yield comparable overall performance, with BGE showing slight advantages under certain configurations. Notably, in the 20-shot, 18

in-memory setup with Eager attention and zero context length, BGE achieves a higher average accuracy (77.80%) compared to LLM (71.70%). The largest gap occurs in the *Subj* task, where BGE reaches 82.85% versus LLM's 50.35%, suggesting that BGE better captures fine-grained semantic representations. Nevertheless, both embedders remain broadly consistent, highlighting ERMAR's adaptability to different embedding backbones.

Ablation on Relevance and Re-ranking. We further evaluate the effect of the Relevance+Reranking mechanism on model perplexity across varying context lengths for *WikiText-103* and *PG-19*. Results are shown 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 on the Relevance+Re-ranking mechanism showing perplexity differences for *WikiText-103* and *PG-19* across context lengths.

The relevance scoring mechanism provides the greatest improvements at shorter context lengths—10.4% at 1K and 7.2% at 2K tokens on *WikiText-103*. The gains taper off beyond 4K tokens and become marginally negative at 32K, suggesting that the computational overhead of ranking may outweigh its benefits when sufficient memory capacity is available. In contrast, *PG-19* exhibits consistent improvements (1.0–9.9%) across all context lengths, indicating that narrative text benefits more from semantic ranking, where long-range dependencies and non-sequential references are frequent.

4 Conclusion

This paper introduced **ERMAR**, a Ranked Memory-Augmented Retrieval framework that enhances long-context language modeling through dynamic relevance scoring and adaptive memory management. By prioritizing semantically relevant information during retrieval, ERMAR improves contextual reasoning and scalability over extended sequences. Experimental results demonstrate that ERMAR achieves up to a **3.2% reduction in perplexity** compared to MemLong and maintains stable performance at **32K-token** contexts. In few-shot in-context learning tasks, ERMAR outperforms baselines by **6.1%** on average in 4-shot and **7.1%** in 20-shot settings, including substantial gains on MPQA (+21%) and Subj (+26%), confirming its effectiveness in retrieval precision and generalization.

ERMAR also exhibits strong computational efficiency, reducing reserved memory by up to **30%** at long contexts and sustaining a throughput of approximately **2.4K tokens/sec** with minimal latency variance. These results establish ERMAR as an efficient and scalable retrieval-augmented framework for long-context modeling. 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 certain limitations. Its reliance on ranked memory structures increases computational overhead compared to standard LLMs, particularly for large-scale retrieval, as discussed in Subsections 3.2.4 and 3.2.5, as well as Appendix A.5. Additionally, performance variations across different task domains indicate the need for further tuning. The framework's effectiveness in real-world, noisy environments also requires further validation.

Acknowledgments

Shoaib Jameel is supported by the Turing's Defence and Security programme through a partnership with the UK government in accordance with the framework agreement between GCHQ & The Alan Turing Institute. This research is partially supported by the Technology Innovation Institute, Abu Dhabi, UAE. Computational experiments were conducted using the Wolfpack cluster, supported by the School of Computer Science and Engineering at UNSW Sydney.

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.

A 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.

A.2 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 2).
- **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.

A.4 Fine-Grained Context Length Analysis

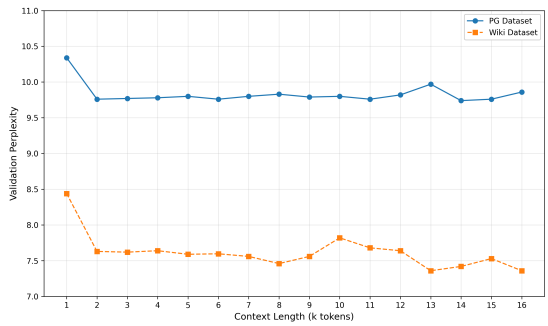

Figure 4: ERMAR perplexity performance across fine-grained context lengths for PG-19 and WikiText-103 datasets. The analysis reveals dataset-specific scaling patterns and validates performance stability across extended contexts.

Figure 4 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 Relevance+Reranker Visual Analysis

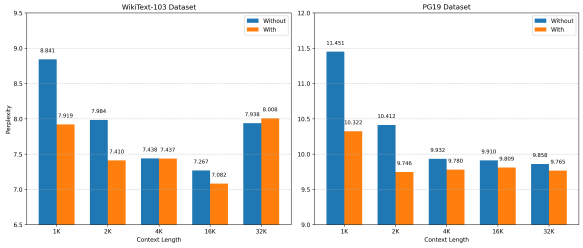

Figure 5: Visualization of Table 9, showing the relative impact of the Relevance+Reranker mechanism across context lengths. Color coding highlights: blue bars represent baseline performance, while orange bars show improvements with our full mechanism(ERMAR).

Figure 5 provides complementary visual evidence for the patterns discussed in Section 3.2.6. The stronger improvements at shorter contexts are visually apparent through the larger differentials between the orange and blue bars, while differences in mechanism effectiveness across datasets are immediately noticeable in the side-by-side comparisons. Additionally, the 32K edge case on WikiText, where the mechanism underperforms, is clearly highlighted in the visualization, underscoring context-length sensitivities of the Relevance+Reranker mechanism.

A.6 Impact of Top-*K* on Performance

We conducted experiments to examine how varying the Top-*K* retrieval parameter affects system performance across different sequence lengths on the *WikiText-103* dataset. The Top-*K* values (1, 4, 8, 16, 32) were tested for sequence lengths of 1K–16K tokens, measuring latency, throughput, memory usage, and perplexity.

Results. Table 10 summarizes latency and memory results across all configurations. Increasing Top-*K* consistently reduces latency per token and

improves throughput, particularly between $K = 1$ and $K = 4$, with diminishing gains beyond $K = 8$. Memory consumption (both peak and reserved) remains constant for a given sequence length, confirming that Top-K selection does not affect model footprint.

Seq Len	TopK	Latency/ Token (ms)	Latency (s)	Throughput (tokens/sec)	Peak Memory (GB)	Reserved Memory (GB)
1024	1	285.38 ± 54.98	0.279	3684.15	7.98	8.19
	4	277.15 ± 60.41	0.271	3810.19	7.98	8.21
	8	270.90 ± 57.72	0.265	3893.17	7.98	8.21
	16	268.60 ± 57.54	0.262	3928.15	7.98	8.21
	32	269.26 ± 56.84	0.263	3915.67	7.98	8.21
2048	1	630.68 ± 49.07	0.308	3264.37	8.46	8.83
	4	556.76 ± 51.17	0.272	3704.12	8.46	8.81
	8	555.35 ± 51.52	0.271	3714.39	8.46	8.81
	16	549.25 ± 50.81	0.268	3755.02	8.46	8.81
	32	549.17 ± 50.93	0.268	3756.11	8.46	8.81
4096	1	1873.54 ± 54.43	0.457	2187.96	9.43	9.89
	4	1804.90 ± 56.60	0.441	2271.53	9.43	9.91
	8	1759.28 ± 53.03	0.430	2330.24	9.43	9.89
	16	1738.93 ± 44.91	0.425	2356.96	9.43	9.90
	32	1719.64 ± 50.91	0.420	2383.85	9.43	9.90
8192	1	4264.45 ± 63.90	0.521	1921.41	11.36	12.16
	4	4116.54 ± 71.41	0.503	1990.60	11.36	12.18
	8	4094.69 ± 79.01	0.500	2001.35	11.36	12.16
	16	4107.60 ± 94.04	0.501	1995.37	11.36	12.16
	32	4098.25 ± 96.92	0.500	2000.00	11.36	12.16
16348	1	9136.22 ± 176.62	0.558	1793.95	15.21	16.69
	4	8965.27 ± 115.76	0.547	1827.79	15.21	16.71
	8	8961.31 ± 55.20	0.547	1828.37	15.21	16.69
	16	8945.37 ± 42.26	0.546	1831.60	15.21	16.70
	32	8843.33 ± 48.01	0.540	1852.75	15.21	16.70

Table 10: Performance metrics across Top-K values and sequence lengths on *WikiText-103*.

Constant Metrics. Table 11 reports memory-per-token and perplexity, which remain stable across Top-K variations. This indicates that increasing retrieval scope affects efficiency but not model quality.

Sequence Length	Memory/Token (MB)	Val Perplexity	Perplexity
1024	7.98	2.13	8.42
2048	4.23	2.03	7.61
4096	2.36	2.03	7.62
8192	1.42	2.05	7.76
16348	0.95	2.05	7.80

Table 11: Constant metrics across Top-K values for each sequence length.

Findings. Increasing Top-K from 1→4 yields the largest latency reduction (e.g., 285.4→277.1 ms at 1K; 9136.2→8965.3 ms at 16K), while further increases provide minimal additional gains. Throughput improvements are proportional, especially for longer sequences (e.g., 2188→2384 tokens/s at 4K). Memory and perplexity remain unaffected. Thus, $K = 4$–8 offers the best trade-off between efficiency and computational cost.

Embedder	In-C, In-M	Dim	MPQA	SST2	SST5	SUBJ	MR
BGE	4,4	512	88.28	93.00	45.96	94.90	97.15
		1024	87.93	91.51	46.32	94.65	97.09
		2048	87.99	91.51	46.14	94.35	97.08
	4,18	512	88.02	92.89	46.14	94.90	97.11
		1024	88.11	91.40	46.32	94.65	97.09
		2048	88.17	91.40	46.14	94.35	97.08
	20,18	512	91.23	94.50	52.59	95.55	97.83
		1024	91.26	93.35	49.41	96.60	97.44
		2048	91.29	93.23	49.68	96.75	97.40
LLM	4,4	512	88.65	93.00	46.32	89.05	97.62
		1024	88.76	92.20	46.32	88.15	96.65
		2048	88.76	92.20	46.32	88.15	96.65
	4,18	512	88.59	93.00	46.32	89.25	97.62
		1024	88.72	92.20	46.32	88.15	96.65
		2048	88.72	92.20	46.32	88.15	96.65
	20,18	512	90.35	92.78	50.32	95.15	98.08
		1024	91.01	93.35	48.14	94.85	98.04
		2048	91.01	93.35	48.14	94.85	98.04

Table 12: Accuracy results for embedding dimensionality experiments across datasets and settings.

A.7 Pruning Strategy Validation

The proportional settings (10% recent, 80% middle, 10% oldest) were initially heuristic. We present systematic experiments validating these ratios across context lengths and text genres.

Setup. We evaluated 11 pruning strategies on WikiText-103 at five context lengths (1K–32K tokens). *Baselines*: **Uniform** (equal priority), **Recency-only**, and **Frequency-only**. *Hybrid strategies* follow the X-Y-Z format (recent%–middle%–oldest%): 0-90-10, 5-80-15, 5-85-10, **10-80-10** (proposed), 10-85-5, 10-90-0, 15-75-10, and 20-70-10. All strategies yield identical perplexity per context length (1K: 8.42; 2K: 7.61; 4K: 7.62; 16K: 7.80; 32K: 7.86), so results below reflect efficiency differences only.

Results. Tables 13 and 14 report efficiency and latency results respectively for all strategies across all context lengths. Table 13 shows that hybrid strategies consistently achieve lower latency per token and higher throughput than baselines, while Table 14 reveals that hybrids also exhibit tighter standard deviations, indicating more stable inference. Table 15 summarizes the best-performing strategy per context length regime.

Findings. (1) Hybrid strategies outperform baselines, yielding 10–45% throughput gains across all lengths (e.g., $+45.4\%$ vs. Uniform at 1K; $+17.0\%$ at 32K). **(2) Optimal ratios shift with context length**: at short contexts (1K–2K), 5-85-10 dominates by minimising the recent slot and maximising the middle; at 4K our proposed

	1K		2K		4K		16K		32K	
Strategy	**ms/tok**	**tok/s**	**ms/tok**	**tok/s**	**ms/tok**	**tok/s**	**ms/tok**	**tok/s**	**ms/tok**	**tok/s**
0-90-10	0.285	3619	0.266	3798	0.374	2676	0.444	2253	0.465	2153
5-80-15	0.259	4023	0.259	3906	0.347	2884	0.441	2269	0.458	2184
5-85-10	0.206	**5069**	0.205	**4946**	0.336	2981	0.416	**2404**	0.414	**2413**
10-80-10 (proposed)	0.262	3981	0.241	4202	0.329	**3044**	0.453	2209	0.424	2359
10-85-5	0.259	3979	0.228	4428	0.344	2913	0.424	2358	0.446	2243
10-90-0	0.308	3512	0.264	3899	0.363	2760	0.452	2213	0.473	2116
15-75-10	0.283	3875	0.224	4631	0.368	2736	0.417	2400	0.435	2297
20-70-10	0.254	4067	0.242	4197	0.346	2894	0.432	2316	0.452	2212
Frequency-only	0.278	3737	0.267	3790	0.372	2688	0.466	2145	0.482	2076
Recency-only	0.264	3921	0.263	3850	0.360	2779	0.453	2206	0.478	2094
Uniform	0.295	3486	0.283	3566	0.375	2673	0.469	2133	0.485	2062

Table 13: Latency per token (ms/tok) and throughput (tok/s) for all pruning strategies across context lengths on WikiText-103. All strategies yield identical perplexity at each context length. Bold denotes the best throughput per context length. Hybrid strategies are separated from baselines by a horizontal rule.

Strategy	**1K (s)**	**2K (s)**	**4K (s)**	**16K (s)**	**32K (s)**
0-90-10	1.39±59.71	544.75±61.74	1533.22±63.16	7273.54±65.57	15222.90±76.59
5-80-15	265.24±67.89	529.87±60.99	1422.65±58.80	7222.95±134.59	15003.15±76.86
5-85-10	210.65±54.01	420.44±58.35	1376.97±64.84	6814.59±52.75	13580.93±41.21
10-80-10 (prop.)	267.94±68.49	492.74±57.10	1348.11±60.15	7419.04±81.91	13894.24±98.75
10-85-5	265.54±57.85	467.92±57.57	1408.47±57.39	6947.99±53.99	14608.47±81.73
10-90-0	315.16±121.40	540.34±110.69	1487.03±66.01	7405.31±50.24	15488.75±77.02
15-75-10	289.29±123.63	458.33±109.72	1506.06±125.70	6826.80±60.86	14264.39±89.72
20-70-10	260.48±59.69	495.15±69.11	1417.23±57.05	7073.92±47.16	14811.33±64.79
Frequency-only	284.33±67.71	545.86±61.15	1524.97±44.72	7641.64±127.35	15787.38±72.64
Recency-only	270.18±62.35	538.19±66.10	1476.49±67.39	7428.95±49.05	15648.46±61.91
Uniform	302.29±63.34	579.82±62.77	1535.08±71.37	7683.59±87.02	15888.88±85.68

Table 14: Total latency (mean ± std, in seconds) for all pruning strategies across context lengths on WikiText-103. Lower values and tighter standard deviations indicate more stable inference. Hybrid strategies are separated from baselines by a horizontal rule.

Context	**Best Strategy**	**Throughput (tok/s)**
1K / 2K	5-85-10	5069 / 4946
4K	10-80-10	3044
16K / 32K	5-85-10	2404 / 2413

Table 15: Best-performing strategy per context length regime.

10-80-10 is best; at long contexts (≥16K), 5-85-10 reasserts via aggressive middle-biased pruning. **(3) Quality is unaffected**: perplexity is identical across all strategies at each context length, confirming that pruning ratios govern efficiency only. **(4) Hybrids are most stable**: hybrid configurations achieve latency variance of ±40–70 ms vs. ±45–127 ms for baselines, making them more reliable in deployment.

A.8 Sensitivity to Embedding Dimensionality in ICL Retrieval

We analyzed how embedding dimensionality influences in-context retrieval performance using both BGE and LLM embedders. Experiments were conducted on *MPQA*, *SST-2*, *SST-5*, *SUBJ*, and *MR* datasets with in-context/in-memory settings of (4, 4), (4, 18), and (20, 18). Embedding dimensions of 512, 1024, and 2048 were evaluated.

Results. Table 12 reports results for both embedders. Overall, reducing the dimension to 512 (via PCA) maintains or slightly improves accuracy across most datasets compared to higher dimensions, suggesting redundancy in high-dimensional embeddings.

Findings. Reducing dimensionality to 512 preserves retrieval accuracy while slightly improving performance on *SST-5* (+3.2 pp) and *MPQA* (+0.3 pp). This indicates that lower dimensions suppress noise and redundancy. Dimensional padding

to 2048 introduces negligible changes (<0.5 pp). Across both embedders, *SUBJ* shows minor degradation at high dimensionality, whereas *MR* remains stable. Overall, 512-dimensional embeddings offer the best trade-off between accuracy and efficiency.