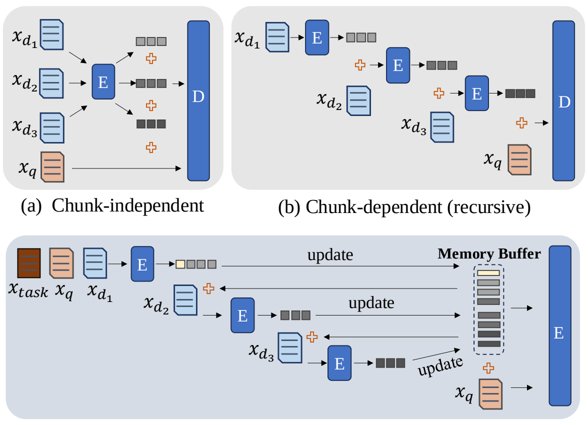
TARMER: A Task-Aware Recursive Prompt Compression Approach with Memory Buffer

Anonymous ACL submission

Abstract

As large language models (LLMs) scale to longer contexts, the quadratic cost of attention poses computational and memory challenges. Segmenting inputs mitigates this but breaks inter-segment dependencies. We propose TARMER, a task-aware recursive prompt compression method with a memory buffer. It jointly models compression and generation as a single forward pass, avoiding intermediate short prompts. Guided by task descriptions and queries, TARMER enhances semantic understanding and task-specific redundancy reduction. Experiments on dialogue, multiple-choice, and out-of-distribution tasks show that TARMER achieves up to 16× compression with minimal performance drop, using a memory buffer with constant space complexity.



(a) Chunk-independent (b) Chunk-dependent (recursive)

(c) Chunk-dependent (task-aware recursive with memory buffer)

Figure 1: Comparison of different soft prompt compression paradigms. (a) Context chunks are compressed independently, (b) Recursive compression incorporating chunk dependencies, (c) The proposed task-aware recursive compression with memory buffer.

1 Introduction

As LLMs gain prominence in language understanding and generation (Li et al., 2024b,c), managing long contexts efficiently has become a key challenge. Transformer-based models (Vaswani, 2017) use self-attention (Bahdanau, 2014) to model token dependencies, but its quadratic complexity with sequence length leads to substantial memory and compute costs. This limits LLMs in in-context learning (Dong et al., 2024; Li et al., 2024a), dialogue (Hosseini-Asl et al., 2020; Savchenko and Savchenko, 2024), and Retrieval-Augmented Generation (RAG) (Lewis et al., 2020).

To optimize the processing of long texts, prior work explores sparse attention (Child et al., 2019; Fu et al., 2022), sliding windows (Beltagy et al., 2020), and extended positional encoding (Chen et al., 2023). As these often require training from scratch, prompt compression has gained attention for its lower training cost. Typical prompt compression methods often segment the context to fit limited window sizes (Li et al., 2023; Chevalier et al., 2023; Yen et al., 2024) (Figure 1a). However, compressing raw input overlooks dynamic redundancy, which can vary across tasks (e.g., summarization vs. QA) and even across queries within the same task.

In response, we propose a *task-aware* prompt compression method that conditions each segment on the task description and current query. This facilitates dynamic redundancy removal by leveraging specific conditions, making the compression more task-specific. For instance, in QA, the model prioritizes question-relevant segments; in dialogue, it highlights contextually salient turns dynamically adapting to the current interaction and minimizing the reliance on redundant prior exchanges.

Furthermore, simply segmenting the original long contexts poses challenges, as global relationships between segments are disrupted, hindering the capture of long-range dependencies. Since the compression *within* each segment suffers from window locality, it fails to eliminate redundancy *across* segments, exacerbating the context fragmentation issue (Dai, 2019). As a result, infrequent but crucial information is overlooked. To tackle this challenge, we propose a condition-aware *recursive* compression paradigm. The core idea is to introduce a recursive mechanism where the compression of each subsequent segment is conditioned on the information provided by the preceding segments. This allows the model to update its global representation using the pre-stored compressed information from earlier segments, thereby overcoming local limitations and reducing *cross-segment* redundancy.

Prior work (Rae et al., 2020) shows that storing past activations in *external memory* is more effective than using state vectors. However, most external memory methods (Chevalier et al., 2023; Kim et al., 2024) use concatenation to update the memory, which causes linear growth in storage costs with segments. In contrast, human working memory operates efficiently despite limited capacity (Baddeley, 1992). Inspired by this, we introduce a fixed-length memory buffer to store global context (Figure 1c), updated via a learned merging function. This enables constant memory usage while preserving compression effectiveness.

During inference, the model is prompted with task descriptions and specific inputs, followed by end-to-end prompt compression and response generation. Our soft prompt compression eliminates intermediate discrete short discrete prompts (Li et al., 2023; Jiang et al., 2023b; Pan et al., 2024), storing the compressed version in hidden representations while optimizing key information retention.

In summary, our contributions are as follows:

- We propose a novel task-aware recursive soft prompt compression paradigm, which effectively identifies dynamic redundancy under varying conditions and removes cross-segment redundancy.

- We introduce a fixed-length memory cache mechanism with a customized merging function that ensures constant space complexity, making it particularly suitable for resource-constrained scenarios.

- Extensive experiments on tasks such as dialogue generation, out-of-distribution recommendation, and in-context learning demonstrate the superiority of our method. Our approach achieves up to a 16x prompt compression ratio without compromising performance.

2 Related Work

2.1 Prompt Compression

Prompt compression generates concise prompts for downstream tasks and falls into two main categories. 1) Discrete hard prompt compression: These methods use a compression model to shorten input text, which is then fed into a separate task model. Entropy-based approaches (e.g., Selective Context (Li et al., 2023)) remove low-information content, while LLMLingua (Jiang et al., 2023b) and LLMLingua-2 (Pan et al., 2024) use LLMs to detect semantic redundancy. 2) Soft prompt compression: These methods optimize continuous embeddings (soft prompts) for end-to-end tasks without separate models (Lester et al., 2021). Distillation-based approaches (Wingate et al., 2022) align distributions but require re-optimization per prompt and lack generality. Compressive Transformer (Rae et al., 2020) compresses activations via convolutions. GIST (Mu et al., 2024) uses input-dependent tokens for attention. ICAE (Ge et al., 2023) adopts an encoder-decoder, while AutoCompressor (Chevalier et al., 2023) recursively summarizes segments. Our method falls into this category, enhancing global redundancy removal via a memory cache that mitigates recursive forgetting.

2.2 Long Context Learning for LLMs

Existing methods for long sequences include expanding the window (Nijkamp et al., 2023), interpolating positional embeddings (Chen et al., 2023), and modifying attention, e.g., sparse (Child et al., 2019) or sliding-window (Beltagy et al., 2020) mechanisms (Peng et al., 2023). Additionally, recent work explores long-term memory for LLMs. Transformer-XL (Dai, 2019) preserves past activations, while MANNs (Meng and Huang, 2018) use external memory matrices to boost capacity. RMT (Bulatov et al., 2022) models memory via read-write tokens, and Memorizing Transformer (Wu et al., 2022) applies memory at the top layer, updating it with kNN. MemoryBank (Zhong et al., 2024) incorporates the Forgetting Curve to enhance long-term memory. In contrast to these approaches, our method does not directly aim to expand the context window of LLMs. Our approach enables efficient access to broader context with lower cost and latency, without retraining or architectural changes, making it complementary to existing techniques.

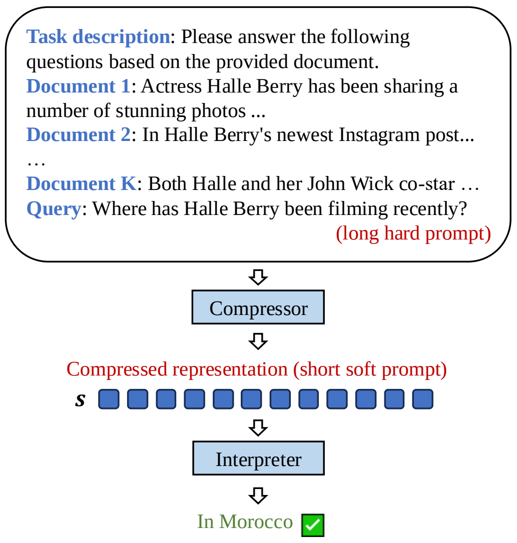

Task description: Please answer the following questions based on the provided document.
Document 1: Actress Halle Berry has been sharing a number of stunning photos ...
Document 2: In Halle Berry's newest Instagram post...
…
Document K: Both Halle and her John Wick co-star …
Query: Where has Halle Berry been filming recently?
(long hard prompt)
Compressor
Compressed representation (short soft prompt)
s
Interpreter
In Morocco ✓

Figure 2: Illustration of the soft prompt compression paradigm. The compressor generates compressed embeddings, which the interpreter uses to create responses. Our compressor and interpreter share parameters and are optimized in an end-to-end manner.

3 Method

We propose TARMER, a parameter-efficient soft prompt compression framework that unifies prompt compression and response generation by optimizing a single language model, as shown in Figure 2.

3.1 Problem Definition

Let $\mathcal{T}$ denote the text space, and given an input query $x_q \in \mathcal{T}$, K context segments $x_d = \{x_{d_1}, x_{d_2}, \ldots, x_{d_K}\} \in \mathcal{T}$, and a target output sequence $y = [y_1, y_2, \ldots, y_n] \in \mathcal{T}$, where $y_1, y_2, \ldots, y_n$ are n tokens of the target response sequence. The context segments can include paragraphs from long documents or a set of retrieved documents. Assume a pre-trained language model $f_\theta : \mathcal{T} \rightarrow \mathbb{R}^+$, which models the probability distribution over the text space $\mathcal{T}$. A typical approach for predicting the output y involves using the full context as:

$$y \sim f_\theta(\cdot|x_d, x_q), \tag{1}$$

where x_d is the context and x_q is the task-specific query. However, this method incurs increasing memory and computational costs as the context length grows. Existing prompt compression frameworks aim to alleviate this issue by compressing lengthy and complex prompts into shorter prompts.

General prompt compression methods (Jiang et al., 2023b; Ge et al., 2023) compress each context segment $x_{d_i} \in x_d$ into smaller summary sequence s_i, which are then assembled into the final compressed prompt sequence $s = [s_1, s_2, \ldots, s_m]$. This compressed prompt s captures the core information of x_d with far fewer tokens. The compressed prompt s is then appended to the input query x_q and provided to the language model f_θ to generate a response $\hat{y} \in \mathcal{T}$ that ought to be similar to the output generated using the full context. When the summary $s \in \mathcal{T}$, i.e., the compressed prompt is discrete text, the corresponding method is hard prompt compression. When s is an embedding representation in the latent space, the method corresponds to soft prompt compression, and our approach falls within this paradigm.

Our method performs global chunk-based recursive compression with the input query x_q and the specific task description x_{task}, enabling the compressed vector s to consider both the current task and the specific query. Our task-aware recursive compression method is described in Section 3.2.

3.2 Task-Aware Recursive Compression

Considering that soft prompts can interpolate between many token embeddings, enabling more abstract representations than a single discrete token (Wingate et al., 2022). In this paper, we adopt the framework of soft prompt compression.

We define two types of memory units: segment memory for the current chunk and sequence memory for the entire sequence of chunks. The segment memory is stored in compressed token embeddings, which originate from the language model's embedding space spanning thousands of dimensions, thus possessing high information transmission capacity. The sequence memory is updated through segment memories and concatenated with subsequent chunks or queries to generate future segment memories or the final response.

Segment Memory. We expect model f_θ to generalize to unseen tasks. Given a new task t, the model should be able to compress the representation based on the task description x_{task}, the input query x_q, and the context x_d to produce accurate responses without any additional training. Therefore, we concatenate x_{task} and x_q in the input to enforce both task-related and query-related constraints:

$$\text{Seg\_mem}_1 = M_\theta(x_{task}, x_q, x_{d_1}, \text{COMP}), \tag{2}$$

where COMP is the compressed token appended to the text sequence, and its corresponding latent representation merges the task description and query information. Subsequent segment memories are generated based on this:

$$\text{Seg\_mem}_i = M_\theta(\text{Seg\_mem}_{i-1}, x_{d_i}, \text{COMP}), \quad (3)$$

where i ranges from 2 to K. The recursive process indicates that the current token only needs to access the state vector from the previous step to complete the current inference step.

Sequence Memory. The above recursive design of segment memory is inspired by the Mamba-based LLM (Gu and Dao, 2023), which is essentially a state space model. However, in Mamba, the state vector at each time step can only store a limited amount of historical context. As the context length increases, memory forgetting occurs, which degrades inference performance. To address this, we propose a *compromise* sequence memory used to cache the historical segment memories. Our key difference from the traditional approach is the introduction of memory buffer strategies, which allow the model to control the sequence memory used by effectively managing the historical context.

A vanilla strategy involves **concatenating** the segment memory with the current memory pool at each step to build the global context:

$$\text{Seq\_mem}_i = \text{Concat}(\text{MemoryPool}_{i-1}, \text{Seg\_mem}_{i-1}), \quad (4)$$

where MemoryPool_{i-1} is the accumulated memory from previous segments and Seg_mem_{i-1} is the newly generated segment memory. The concatenation strategy, while simple, leads to a growing memory pool. As K increases, this can result in substantial memory overhead.

To address this, we introduce the **memory buffer** strategy, which aims to limit the growth of memory size while maintaining essential context information. In this strategy, the sequence memory is not simply concatenated but updated by a memory editing model E_θ, which adjusts the memory pool by selectively retaining critical information from past segment memories. The updated sequence memory is denoted as:

$$\text{Seq\_mem}_i = E_\theta(\text{MemoryPool}_{i-1}, \text{Seg\_mem}_{i-1}), \quad (5)$$

where E_θ is a memory editing model with two transformer blocks that compute self-attention over the segment memory and the global memory pool. The editing model ensures that the memory size does not increase with K and remains bounded by a fixed length. This memory buffer design allows the model to focus on the most relevant information across the entire sequence without the need to store all historical segments. By adopting the memory buffer strategy, we achieve a significant reduction in memory usage, especially when handling large amounts of context, while still retaining sufficient historical context for accurate responses.

Since the sequence memory contains the key information of all chunks, during inference, the final response sequence can be generated using the global sequence memory:

$$\hat{y} = f_\theta(\text{Seq\_mem}_K, x_q). \quad (6)$$

3.3 Optimization

Optimization Function. In this work, we do not introduce additional decoders, as in other soft prompt methods such as ICAE (Ge et al., 2023) or CEPE (Yen et al., 2024). Instead, we directly optimize the encoder in an end-to-end manner for target response. Unlike optimizing all parameters of the encoder, we only optimize the LoRA adapters and the embedding layer of the COMP tokens.

Given the final sequence memory Seq_mem_K and the input query x_q, we compute the probability distribution over the next token in the sequence using the pre-trained language model M_θ. The objective is to minimize the cross-entropy loss between the predicted output and the ground truth target response. Let $\hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n]$ represent the predicted output sequence, and $y = [y_1, y_2, \ldots, y_n]$ represent the ground truth target sequence. The probability distribution over the next token, conditioned on the sequence memory Seq_mem_K and query x_q, is given by:

$$P(\hat{y}_n | \text{Seq\_mem}_K, x_q) = f_\theta(\hat{y}_n | \text{Seq\_mem}_K, x_q), \quad (7)$$

where $n = 1, 2, \ldots, N$ and N is the total number of tokens. The cross-entropy loss at each position n is computed as:

$$\mathcal{L}_n = -\log P(y_n | \text{Seq\_mem}_K, x_q). \quad (8)$$

y_n is the true token at position n, and $P(y_n | \text{Seq\_mem}_K, x_q)$ is the predicted probability. The total cross-entropy loss for the response sequence is the sum of the individual token losses:

$$\mathcal{L}_{\text{CE}} = \sum_{n=1}^{N} \mathcal{L}_n = -\sum_{n=1}^{N} \log P(y_n | \text{Seq\_mem}_K, x_q). \quad (9)$$

The goal of optimization is to minimize the total cross-entropy loss $\mathcal{L}_{\text{CE}}$, which directly drives the model to refine its predictions of the target response based on the joint optimization of the recursive compression process, task-awareness, and sequence memory. Leveraging soft prompts, the loss is computed end-to-end, eliminating the need for intermediate discrete prompt tokens or the additional training of separate decoders. In practice, only the LoRA adapters, COMP token embeddings, and lightweight editor E are updated, reducing the parameter space and enhancing learning efficiency.

Memory Analysis. We analyze the two proposed memory strategies: the concatenation-based strategy TARMER-C and the fixed-length memory buffer strategy TARMER-B. We also compare them with the traditional full-context mode.

Traditional **long-context** methods store the relationships between every token in the entire context within a large attention matrix. The memory requirement for this approach grows quadratically with the total context length: $M_{\text{full}} = \mathcal{O}((K \times l_c)^2)$ where l_c is the length of each segment, and K denotes the total number of segments. The concatenation-based strategy, **TARMER-C**, stores the activation of each segment in external memory, updating the memory as new segments are processed. The memory requirement grows linearly with the number of segments K: $M_{\text{concat}} = \mathcal{O}(K \times l_c)$. In contrast, the fixed-length memory buffer strategy, **TARMER-B**, stores global sequence within a fixed-size memory buffer. Instead of concatenating, the buffer retains a fixed-size vector and updates it as a new global memory. The memory complexity is: $M_{\text{buffer}} = \mathcal{O}(1)$.

The concatenation-based TARMER-C incurs linear memory growth with segment count K. As K increases significantly, adding each new segment potentially leads to memory pressure. On the other hand, the fixed-length memory buffer provides a significant improvement by maintaining constant memory usage and offers a more scalable solution through adjusting the fixed memory size.

4 Experiments

In this section, we provide empirical validation of the proposed method. Through comparisons with existing soft prompt compression approaches, we demonstrate the effectiveness of our method. In Section 4.3, we further confirm its adaptability through ablation studies and additional analyses.

Methods	MetaICL	LaMP	DailyDialog
No context	51.7	69.5	9.85
Full context	69.9	83.1	5.84
GIST	59.7	79.6	7.68
Compressive	67.6	81.6	6.52
CCM	68.9	84.0	6.17
TARMER-C	**70.1**	**85.5**	5.81
TARMER-B	69.8	85.3	**5.67**

Table 1: Comparison with state-of-the-art soft prompt compression methods. Accuracy (%) on MetaICL and LaMP datasets, and perplexity(↓) on DailyDialog dataset.

4.1 Experimental Setup

Implementation Details. In this paper, our objective is to train LLMs end-to-end to perform both long text compression and response generation for downstream tasks. Given that decoder-only models outperform encoder-decoder models in prompt compression (Mu et al., 2024), we use LLaMA-2 (Touvron et al., 2023) and Mistral (Jiang et al., 2023a) as the underlying backbones. For model training, we set the length of the compression token <COMP>to 16 by default, with the length of the memory buffer set to 128. For each sample, we prepend the task description and query to the beginning of the first chunk. We leverage LoRA for efficient parameter fine-tuning, rather than full-parameter training, and apply the same LoRA (Hu et al., 2021) configuration and training protocol across all methods. We set the rank r of the LoRA parameters to 8. Additionally, FlashAttention (Dao et al., 2022) is used to accelerate the baseline model experiments. All training is conducted on NVIDIA A100 devices.

Datasets We evaluate our method using four datasets: MetaICL (Min et al., 2021), LaMP (Salemi et al., 2023), DailyDialog (Li et al., 2017), and the Prompt-with-Context (PwC) dataset (Ge et al., 2023). First, MetaICL is a multi-task context learning dataset designed to address tasks that were unseen during training, comprising 61 training tasks and 26 unseen test tasks. The evaluation metric used is the accuracy of multiple-choice questions. The LaMP dataset generates personalized recommendations using user profiles, and we evaluate the multiple-choice accuracy for new users that were not encountered during training. The DailyDialog dataset evaluates performance in dialogue scenarios, containing se-

TARMER	Methods	Judgement (%) Win	Lose	Tie
TARMER-C (LLaMA)	ICAE(k=256)	78.5	10.3	11.2
	CCM-concat	73.4	13.1	13.5
	Full context	33.7	31.7	34.6
TARMER-B (LLaMA)	ICAE(k=256)	70.9	11.6	17.5
	CCM-concat	66.8	25.1	8.1
	Full context	29.6	36.2	34.2
TARMER-C (Mistral)	ICAE(k=256)	81.0	8.2	10.8
	CCM-concat	85.8	5.3	8.9
	Full context	56.0	22.5	21.5
TARMER-B (Mistral)	ICAE(k=256)	77.7	13.4	8.9
	CCM-concat	79.0	19.2	1.8
	Full context	40.3	28.6	31.1

Table 2: Evaluation using GPT-4 to compare the response quality of our TARMER method with that of other baseline methods.

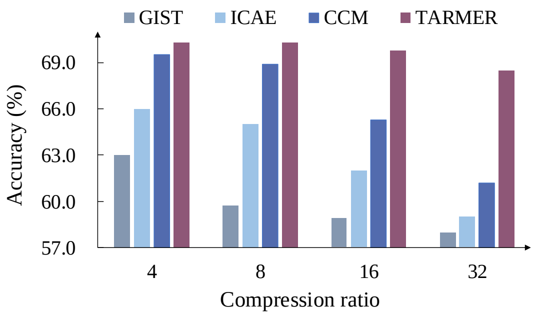


Figure 3: Accuracy (%) on MetaICL at different compression ratios.

quences of everyday conversations. We measure model performance using perplexity on actual dialogues. The PwC dataset is specifically designed for instruction-following tasks that include both context and prompts, containing thousands of samples. In the case of PwC, we do not rely on a specific metric but instead assess the performance of generated sequences using GPT-based evaluation.

4.2 Benchmark Comparison

Metric Comparisons. As show in Table 1, we present performance comparisons across the MetaICL, LaMP, and DailyDialog datasets. Based on existing open-source code, we compared the soft prompt compression methods GIST and CCM, both of which aim to compress attention hidden states. For CCM, we report the performance of the better-performing concat version. To ensure a fair comparison, we set the same compression ratio for all methods. TARMER-C represents the proposed method using a concatenation strategy with a complexity of $O(n)$, which is consistent with other comparison methods. TARMER-B represents the proposed method using a fixed-length memory buffer with a complexity of $O(1)$. For MetaICL and LaMP, document segments evenly contain task-related or user-related information, and the redundancy between documents is minimal. The concat compression strategy often better preserves key information. However, in DailyDialog dataset, the contextual information conveyed in later dialogues becomes increasingly important. In such scenarios, where stricter information filtering is required, our memory buffer-based approach significantly outperforms the simple concat-based compression strategy. This demonstrates that as the redundancy of contextual information increases, the effectiveness of the memory buffer becomes more pronounced.

Comparisons with GPT Evaluations. To evaluate the instruction-following performance on downstream tasks, we not only use fixed metrics but also introduce GPT-based scoring for the generated outputs, as shown in Table 2. We fine-tune using the PwC (Ge et al., 2023) instruction dataset, which consists of 240k (context, query, response) samples for training and 18k samples for testing. Our training and testing setup follows the ICAE (Ge et al., 2023) method. We use Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a) and LLaMA-2-7B-chat (Touvron et al., 2023) as our backbone language models. We compare the response quality of our TARMER method (first column) with several existing baseline methods (second column) using GPT-4 (Achiam et al., 2023) to assess which approach performs better or whether they are comparable. To ensure a fair comparison, all baseline methods utilize the same backbone model, and both ICAE and CCM-concat (Kim et al., 2024) store information from different chunks using memory slots with linear complexity.

In Table 2, TARMER-C demonstrates superior performance over ICAE and CCM-concat, and they use the same memory footprint. TARMER-C gains the win rates of 81.0% and 85.8% for Mistral. Compared to modeling the full context, TARMER-C achieves a win-tie rate of 77.5%. TARMER-B, which introduces a fixed-size memory buffer, balances memory usage and performance, achieving average win rates of 74.3% and 72.9%, surpassing ICAE and CCM.

Methods	No context	Full context	AutoComp	AutoComp-FT	CCM-C	CCM-M	TARMER-C	TARMER-B-64	TARMER-B-128
Accuracy	41.4	54.2	48.1	50.9	53.5	52.3	54.4	53.4	53.7
Memory	31	394	156	156	111	41	128	39	77

Table 3: Comparison of accuracy (%) and peak KV memory (MB) with SOTA recursive compression methods using the OPT-2.7B model.

Memory Efficiency. In Figure 3, we evaluate the performance of various soft prompt compression methods at different compression ratios on the MetaICL dataset. The methods considered for comparison are GIST (Mu et al., 2024), ICAE (Ge et al., 2023), and CCM (Kim et al., 2024), with TARMER representing our approach that incorporates a memory cache mechanism for efficient compression. At a compression ratio of 4x, TARMER achieves the highest performance (70.3%), outperforming other methods. As the compression ratio increases to 32x, TARMER's performance remains stable at 68.5%, surpassing the performance of most comparative methods at an 8x compression ratio. This indicates that the proposed memory buffer mechanism is robust, effectively mitigating information loss even at higher compression levels. In contrast, other methods show varying degrees of performance degradation as the compression ratio increases. GIST, for instance, drops from 63.0% at a ratio of 4x to 58.9% at a ratio of 16x, indicating a substantial loss in performance as more information is compressed. Similarly, ICAE and CCM exhibit a steady decline in performance as the compression ratio increases, with ICAE reaching only 62.0% at a ratio of 16x and CCM reaching 61.2% at the same compression level. TARMER consistently outperforms these methods across all compression ratios, particularly at higher compression levels (16x and 32x), where it significantly outperforms the other methods. This suggests that the memory buffer mechanism in TARMER effectively mitigates the trade-off between compression and information retention, enabling it to achieve superior performance without relying on excessive memory usage.

To ensure a fair comparison, we selected similar recurrent baseline methods (Chevalier et al., 2023; Kim et al., 2024). The AutoCompressor model was pre-trained on the Pile dataset (Gao et al., 2020) and fine-tuned with MetaICL to obtain AutoCompressor-FT. As shown in Table 3, we compare accuracy and memory usage on the OPT-2.7B model (Zhang et al., 2022). Our method outperforms others in both performance and memory efficiency. TARMER-C, based on hidden vector

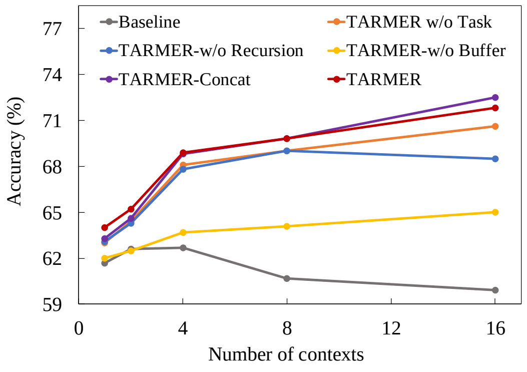


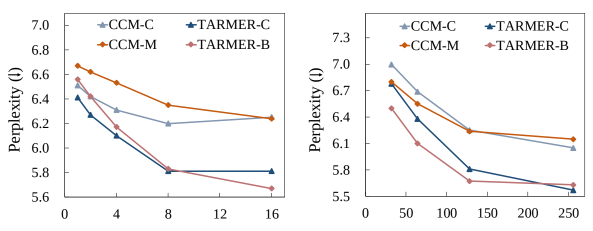
Figure 4: Ablation study on the MetaICL with LLaMA.

concatenation, requires only 128MB of memory, lower than most methods. For methods with comparable performance, memory usage is typically 2-5 times higher than TARMER. Further, TARMER-B's memory consumption can be adjusted based on buffer size. While slightly lower in accuracy than using the full context, TARMER-B still outperforms other compression methods. In contrast to CCM-M (Kim et al., 2024), which sacrifices accuracy, TARMER-B balances memory efficiency and performance better.

4.3 Ablation Studies and Additional Analyses

Ablation Studies. We conducted a comprehensive ablation study to evaluate the effectiveness of our task-aware recursive prompt compression framework with memory buffer. We assessed accuracy on the MetaICL (Min et al., 2021) dataset using LLaMA-7B, as shown in Figure 4. To investigate the impact of different components on TARMER, we introduced the following variants for ablation:

1) "Baseline": The context is directly split into multiple chunks, and the embeddings of these chunks are concatenated to generate the response. 2) "TARMER w/o Task": Task descriptions and queries are not appended when encoding the context chunks. 3) "TARMER w/o Recursion": Previous chunk representations are not used when encoding subsequent chunks. 4) "TARMER w/o Buffer": The memory buffer is replaced with a simple averaging method to update sequence memory, both maintaining constant space complexity. 5)



(a) <COMP>token number (b) Length of memory buffer

Figure 5: (a) The effect of the local segment <COMP>token number. (b) The effect of the global sequence memory buffer. We conduct comparisons with strong baseline CCM.

Input (Batch × Length)	Method	Compress Time	Decode Time	End-to-end Time
4×1024	LLM	0.0	9.1	9.1
	TARMER	1.9	2.6	4.5 (**2.0**×)
4×512	LLM	0.0	7.2	7.2
	TARMER	0.7	2.6	3.3 (**2.2**×)
8×512	LLM	0.0	9.6	9.6
	TARMER	1.2	2.6	4.0 (**2.4**×)

Table 4: Latency comparison of original LLM and our TARMER.

"TARMER-Concat": The recursive chunk memories are concatenated to replace the fixed-length memory buffer, resulting in linear space complexity. 6) "TARMER": The proposed full model with memory buffer.

In Figure 4, "TARMER w/o Recursion" fails to remove redundant chunks due to the lack of inter-chunk relationships, causing critical information loss and performance degradation as context lengthens. "TARMER w/o Buffer" dilutes critical information through averaging, limiting performance gains with longer contexts. "TARMER-Concat" matches or outperforms the full model in accuracy at longer lengths but incurs linear memory consumption. In contrast, TARMER optimally balances performance and memory efficiency.

Analysis of Compressed Memory Length. In the segment memory section, we introduce the <COMP>token to store the compressed content of segments. In the sequence memory strategy, we employ a fixed-length vector MemoryPool to store the global compressed embeddings. The effects of both strategies on the final performance are analyzed in the figure. Initially, we vary the length of the <COMP>token while fixing the global memory length at 128. This implies that as the length of the <COMP>token increases, the number of segments that can be stored in the global memory decreases, as shown in Figure 5(a). Although increasing the number of <COMP>tokens allows more information to be retained, it does not necessarily lead to improved response quality in the concat strategy, as excessive redundant information can introduce unnecessary interference. Next, we fix the number of <COMP>tokens at 16 and vary global memory length, as shown in Figure 5(b). The performance of concat strategy is highly sensitive to memory size. When the cache is smaller than 128, it performs worse than the fixed-length memory buffer. Overall, the proposed TARMER strategy consistently outperforms the strong baseline CCM (Kim et al., 2024) at the same compression rate, as seen in Figure 3.

Analysis of inference efficiency. As depicted in Table 4, we test different latencies with fixed generation lengths for a fair comparison. Results show that decoding process is the main bottleneck in latency. Our compression method reduces decoding complexity with minimal overhead, enabling more than 2x speedup in end-to-end inference. In scenarios, *e.g.*, book and legal document analysis, pre-caching the compressed global sequence memory achieves over 3.5x speedup in decoding. Although TARMER does not alter the quadratic time complexity introduced by the self-attention mechanism, its compression and caching strategies accelerate the inference.

5 Conclusion

To address the computational and memory bottlenecks encountered in large language models when processing long input contexts, this paper proposes a task-aware recursive prompt compression paradigm with a fixed-length hidden vector memory cache. Our approach captures long-range dependencies while controlling memory consumption by dynamically identifying task-relevant redundancy and removing cross-segment redundancy. Moreover, the global memory cache mechanism in fixed size constrains memory consumption, preventing it from scaling with the input length.Extensive experiments on different benchmarks demonstrate up to 16× prompt compression ratio while maintaining performance and improving long-context learning efficiency. Future research could explore the adaptability to various tasks and model architectures, as well as integration with other long-sequence modeling techniques across domains.

6 Limitations

The proposed fixed-memory buffer strategy requires manual tuning of the memory buffer's vector dimension, which may introduce additional effort during deployment. However, we observe that this parameter generalizes well across different datasets. As a result, it can be set to a fixed value to achieve strong performance without per-task tuning.

References

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

Alan Baddeley. 1992. Working memory. *Science*, 255(5044):556–559.

Dzmitry Bahdanau. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

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

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Zihang Dai. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, and 1 others. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128.

Zhihong Fu, Zehua Fu, Qingjie Liu, Wenrui Cai, and Yunhong Wang. 2022. Sparsett: Visual tracking with sparse transformers. *arXiv preprint arXiv:2205.03776*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, and 1 others. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue. *Advances in Neural Information Processing Systems*, 33:20179–20191.

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*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023a. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. 2024. Compressed context memory for online language model interaction. In *ICLR*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Guozheng Li, Peng Wang, Jiajun Liu, Yikai Guo, Ke Ji, Ziyu Shang, and Zijie Xu. 2024a. Meta in-context learning makes large language models better zero and few-shot relation extractors. *arXiv preprint arXiv:2404.17807*.

Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024b. Pre-trained language models for text generation: A survey. *ACM Computing Surveys*, 56(9):1–39.

Qian Li, Zhuo Chen, Cheng Ji, Shiqi Jiang, and Jianxin Li. 2024c. Llm-based multi-level knowledge generation for few-shot knowledge graph completion. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, volume 271494703.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957*.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Lian Meng and Minlie Huang. 2018. Dialogue intent classification with long short-term memory networks. In *Natural Language Processing and Chinese Computing: 6th CCF International Conference, NLPCC 2017, Dalian, China, November 8–12, 2017, Proceedings 6*, pages 42–50. Springer.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*.

Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Erik Nijkamp, Tian Xie, Hiroaki Hayashi, Bo Pang, Congying Xia, Chen Xing, Jesse Vig, Semih Yavuz, Philippe Laban, Ben Krause, and 1 others. 2023. Xgen-7b technical report. *arXiv preprint arXiv:2309.03450*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

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

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2023. Lamp: When large language models meet personalization. *arXiv preprint arXiv:2304.11406*.

Andrey V Savchenko and Lyudmila V Savchenko. 2024. Inside out: emotional multiagent multimodal dialogue systems. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 8784–8788.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. *arXiv preprint arXiv:2210.03162*.

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

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

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