Pretraining Context Compressor for Large Language Models with Embedding-Based Memory

Yuhong Dai[1], **Jianxun Lian**[2], **Yitian Huang**[1], **Wei Zhang**[1], **Mingyang Zhou**[1], **Mingqi Wu**[3], **Xing Xie**[2], **Hao Liao**[1]
[1]College of Computer Science and Software Engineering, Shenzhen University, China
[2]Microsoft Research Asia
[3]Microsoft Gaming
jianxun.lian@outlook.com, haoliao@szu.edu.cn

Abstract

Efficient processing of long contexts in large language models (LLMs) is essential for real-world applications like retrieval-augmented generation and in-context learning, especially in resource-constrained environments such as edge computing. This paper explores the embedding-based context compression to reduce inference costs while preserving the downstream LLM configurations. We propose a decoupled compressor-LLM framework, pretrained on text reconstruction and completion tasks, designed to effectively preserve essential contextual information within condensed embedding representations. Our extensive experiments investigate pretraining, model configurations, compression rates, efficiency across tasks, and adaptability to various LLMs. Results demonstrate that our approach outperforms competitive baselines in three domains and across eight datasets while being adaptable to different downstream LLMs. We find that thorough pretraining and carefully selected compression rates, such as 4x and 16x, enable a lightweight compressor to achieve a good balance between accuracy and speed. These findings underscore the potential of embedding-based compression to enhance LLM efficiency and motivate further research in this area.

1 Introduction

Processing long context as input in large language models (LLMs) is essential for numerous real-world tasks, such as retrieval-augmented generation (RAG) (Gao et al., 2023), in-context learning (Li et al., 2024a) scenarios with extensive content in each demonstration, and life-long AI companions (Zhong et al., 2024) requiring the preservation of past interactions for a consistent experience. Consequently, enabling LLMs with long context processing capabilities (Chen et al., 2023b; Ding et al., 2024) has become a significant trend in newly released models. However, inference with long context poses severe challenges, as the computational cost increases quadratically with the sequence length (as illustrated in Figure A1), which is particularly unfriendly for low-resource devices such as consumer endpoints.

To mitigate these costs without altering LLM architecture, explicit and implicit compression methods have been proposed. Explicit compression methods, such as Selective Context (Li et al., 2023) and LLMLingua-2 (Pan et al., 2024), compact the context by removing redundant or unimportant tokens from the input. Implicit compression methods, such as AutoCompressor (Chevalier et al., 2023), xRAG (Cheng et al., 2024), and ICAE (Ge et al., 2024), compress the input into several dense embedding vectors (aka memory slots). This approach is inspired by the idea that embedding representations act as a new type of compact information modality, conveying denser information than explicit text tokens.

In this paper, we focus on implicit compression, as it has the potential to achieve higher compression rates, leading to more efficient LLM inference. While there are several pioneering works in this direction, we observe that some fundamental questions remain understudied. For instance, can we develop a universal compressor that benefits various downstream LLMs? Can we decouple the compressor from the downstream LLMs, allowing the compressor architecture to be much more lightweight while still maintaining high performance? Additionally, how does the performance change with varying compression rates? Addressing these questions, our research aims to provide insights into designing and optimizing implicit compression methods for efficient LLMs.

We propose a context compression architecture, as illustrated in Figure 1. Our design principle is to decouple the context compressor from the downstream LLM, ensuring that the compressor can be lighter than the LLM. The compressor and

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28715–28732

the LLM generator are connected via a converter, which projects compressed memory slots into compatible vectors for the downstream LLM generator. We pretrain the context compressor using two tasks: the reconstruction task and the language completion task. The goal of these tasks is to enable the compressed slots to effectively memorize useful information from the context, thereby supporting high-quality language generation. We conduct thorough experiments, examining key aspects of the context compressor, such as pretraining tasks, model and data size, compression rate, and efficiency in various downstream tasks. These experiments lead to the following key findings:

Pretraining: Thorough pretraining of the compressor model is critical for achieving optimal performance. Both content reconstruction and language completion tasks are important for generating effective memory slots.

Suggested Compression Rates: 4x and 16x compression rates are recommended for current applications. A 4x compression achieves near-perfect content reconstruction, while 16x offers significant speed gains with minimal information loss.

Higher Compression Rates: Rates like 128x and 256x can still condense useful information but result in higher information loss, highlighted by the challenges in compressor pretraining.

Model Size: Larger backbone models for compressors generally produce higher quality memory slots, indicating the untapped capacity of memory slots.

Scaling to Longer Contexts: To scale to longer context compression with multiple memory slots, it is important to wrap each segment of memory slots with special tokens to clearly delineate memory boundaries.

These findings suggest that embedding-based context compression holds significant potential for improving the efficiency of LLM inference, though its full capabilities have yet to be fully explored. We hope this paper serves as a foundation and motivates further research on pretraining universal context compressors for LLMs. To summarize, the main contributions of this paper are:

- We conduct empirical studies on context compression for efficient LLM inference, utilizing a decoupled compressor-LLM framework and two pretraining tasks. We emphasize that embedding-based compression is a promising and underexplored direction in current literature.
- We conduct a comprehensive investigation into the impact of pretraining and other key configurations, such as compression rate and memory boundary tokens, on the effectiveness of context compression. Based on our findings, we provide key insights for best practices in this area.
- Our method outperforms competitive baselines across three different domains and eight datasets. Additionally, we demonstrate the high adaptability of our method to various downstream LLMs. Source code are released at https://aka.ms/memorycompressor.

2 Methodology

2.1 Problem Definition

This paper focuses on implicit context compression. Given an input sequence $\mathbf{x} = (x_1, \ldots, x_t)$ of t tokens, a compressor model converts $\mathbf{x}$ into a dense memory representation:

$$\widetilde{\mathbf{h}} = f_{comp}(x_1, \ldots, x_t) \tag{1}$$

where $\widetilde{\mathbf{h}} = (\mathbf{h}_1, \ldots, \mathbf{h}_m)$ is a concatenation of m embedding vectors, and the compression rate is $r = t/m$. The dense memory can then substitute the original context $\mathbf{x}$ for downstream tasks in an LLM for text generation:

$$f_{llm} : (\widetilde{\mathbf{h}},\ prompt) \rightarrow \mathcal{X} \tag{2}$$

Our approach involves a two-stage training process that includes both pretraining and fine-tuning of the compressor, while keeping the downstream LLM frozen. An overview of the framework is presented in Figure 1.

2.2 Model Architecture

To enhance adaptability to different LLMs, our compressor consists of two components: an encoder and a converter. The encoder condenses the context into embedding representations, with the embedding dimension determined by the encoder's backbone model. These representations are then processed by the converter, which scales their dimensions and adjust the embedding semantics. This ensures that the memory vectors are compatible with a wide range of downstream LLMs, allowing for more efficient and flexible inference across diverse models and tasks.

Memory Encoder. The memory encoder transforms textual input into high-information-density memory embeddings. As mainstream LLMs have

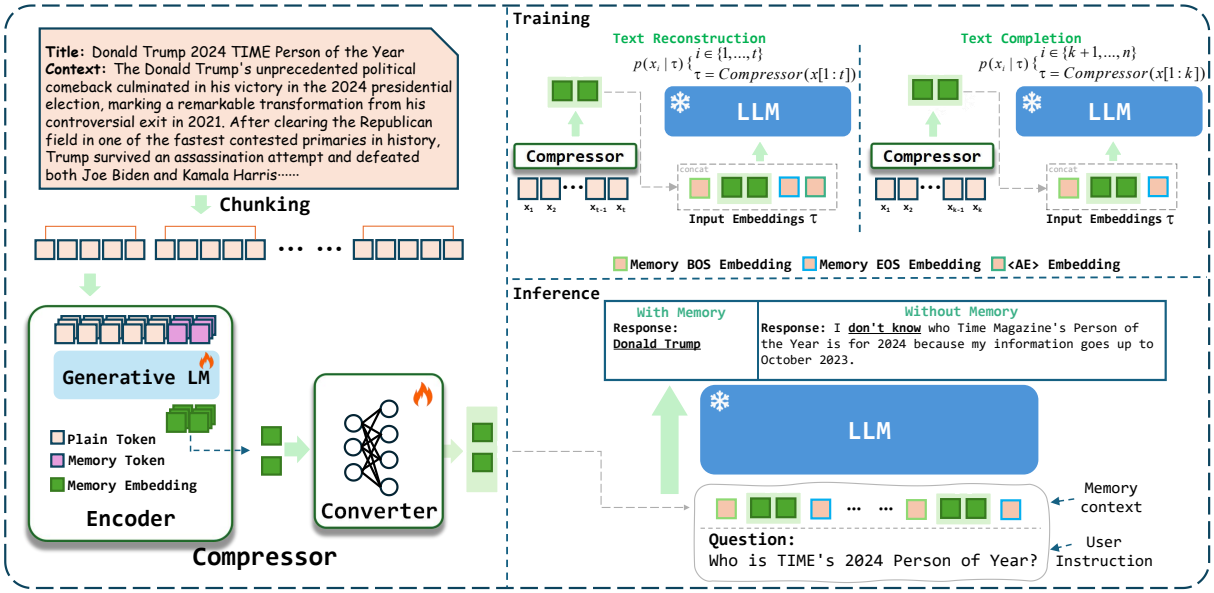


Figure 1: Overview of our PCC Framework. Given a long context, the compressor condenses the content into embedding-based memory slots for fast and accurate inference with an downstream LLM.

increasingly adopted the decoder-only transformer structure in recent years, we use this structure as the backbone in experiments, utilizing two variants: GPT2-Large[1] and Llama-3-8B-Instruct[2]. Specifically, given a sequence of input tokens to be compressed, $(x_1, \ldots, x_t)$, we first append m special memory tokens $(< mem_1 >, ..., < mem_m >)$ to the end. We then take the hidden representations of these memory tokens from the final transformer layer as the encoded memory slots. In our experiments, t is set to 256 as we explore the maximum compression rate of 256x. If the original context is longer than t, it will be divided into multiple segments, each generating memory slots independently.

Memory Converter. The converter acts as an intermediary to adapt the memory embeddings, ensuring compatibility with the target LLM decoder. It addresses potential mismatches in both the embedding dimensions and the semantics produced by the memory encoder, aligning them with the requirements of the downstream LLM decoder. Additionally, the converter facilitates efficient adaptability to various downstream LLMs without large-scale pre-training. This strategy utilizes the pre-trained knowledge embedded in the compressor while enabling efficient adaptation to new decoders. Specifically, the converter is a two-layer MLP, with RMSNorm (Zhang and Sennrich, 2019) applied to the memory embeddings and GELU (Hendrycks and Gimpel, 2016) activations in the first hidden layer.

2.3 Pre-training

Unlike other scenarios such as dense retrieval (Xiao et al., 2022) or generative language modeling (Radford et al., 2018), where pre-training tasks typically involve either token reconstruction or next token prediction, compressed memories play a unique role. On one hand, we want the memories to recall the original content, and on the other hand, we want them to assist in the generation of future tokens. Therefore, we use both types of pre-training tasks: Auto-Encoding for text reconstruction and Auto-Regression for text completion. Specifically, the compressor first compresses the original lengthy context $\mathbf{x}$ into memory slots $\widetilde{\mathbf{h}}$. To distinguish the boundaries of memory slots from different context segments (as mentioned in the *Memory Encoder* Section, lengthy context will be divided into segments), a pair of special tokens <MEM> and </MEM>, which indicate the beginning and end of a memory representation, are used to wrap each memory slot. Containing memory slots in the input prompt, an LLM performs the following pre-training tasks:

Text Completion: The purpose of text completion is to allow memory representation to extract information from the original text that is beneficial to LLMs. Using the memory representation $\widetilde{\mathbf{h}}$ of

[1] https://huggingface.co/openai-community/gpt2-large

[2] https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

the input sequence prefix $(x_1, ...x_{k-1})$, the LLM completes the rest part of sequence:

$$\mathcal{L}_{TC} = -\frac{1}{n-k}\sum_{i=k+1}^{n} log\ f_{llm}(x_i|\ \widetilde{\mathbf{h}}, x_k, ...x_{i-1}) \tag{3}$$

Text Reconstruction: Instead of using a prompt like "Please repeat the sentence:" to instruct the LLM to perform the text reconstruction task, we follow (Ge et al., 2024) and use a new token <AE> after the memory $\widetilde{\mathbf{h}}$ to prompt the LLM to conduct text reconstruction:

$$\mathcal{L}_{TR} = -\frac{1}{n}\sum_{i=1}^{n} log\ f_{llm}(x_i|\ \widetilde{\mathbf{h}}, \texttt{<AE>}, x_1, ...x_{i-1}) \tag{4}$$

The final loss function, $\mathcal{L}$, is defined as:

$$\mathcal{L} = \lambda\mathcal{L}_{TC} + (1-\lambda)\mathcal{L}_{TR} \tag{5}$$

We observe that the difficulty of text reconstruction from memories increases with the length of the original context. To facilitate training, we propose a two-step strategy. In the first stage, we train the model on a warm-up dataset containing shorter texts and utilizing a lower compression rate. Specifically, texts are divided into fixed-length sequences of 128 tokens and a 4x compression rate is applied. This configuration is used solely for the text reconstruction task and utilize 32M tokens for training. In the second stage, training is conducted on a larger dataset with 5B tokens with extended text segments of 256 tokens. Both text completion and text reconstruction tasks are carried out concurrently, with the weighting coefficient λ set to 0.5. Since pre-training is crucial for achieving a high-quality context compressor, we denote our method as **PCC**, short for "Pre-training Context Compressor."

2.4 Fine-tuning

While pre-training enables the compressor to condense information in a general manner, different specific scenarios, such as RAG-based QA or in-context learning, may require slightly different memorization patterns. To capture this nuance, we conduct domain-level fine-tuning on the compressor. Typically, the fine-tuning stage requires only a small amount of domain-specific data to achieve superior performance. Here, "domain-level" indicates that we do not fine-tune the compressor on a dataset-by-dataset basis. Instead, we group datasets by their application type (such as RAG) and fine-tune on only one of the datasets within the same domain to observe whether the compressor can generalize to other datasets within this domain.

Stage	Source	Train	Test
Pre-training	FineWeb	19,167,479	576
Fine-tuning	SQuAD	86,821	5,928
	GSM8K	6,725	748
	HPD	987	110

Table 1: Basic Statistics on the number of data samples of the Datasets

3 Experiments

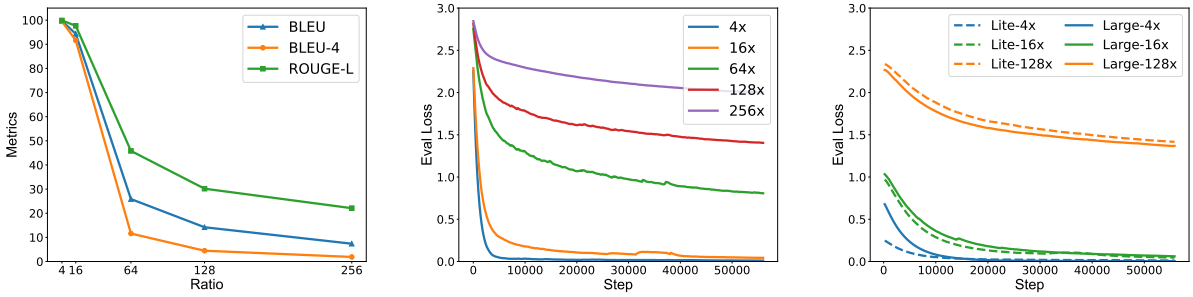
3.1 Experimental Setup

3.1.1 Datasets

This study leverages multiple datasets, including FineWeb (Penedo et al., 2024), SQuAD (Rajpurkar et al., 2018), and GSM8K (Cobbe et al., 2021), to support training and evaluation. A detailed summary of the datasets utilized at different training stages is provided in Table 1. During the pre-training phase, the FineWeb dataset is employed. We utilize 5 billion tokens from the FineWeb. For the fine-tuning phase, three datasets are incorporated: SQuAD, with 86,821 training examples and 5,928 test examples; GSM8K, consisting of 6,725 training examples and 748 test examples; and HPD (Harry Potter Dialogue dataset) (Chen et al., 2023a), containing 987 training examples and 110 test examples. The training sets of SQuAD, GSM8K and HPD are employed as the datasets for the QA tasks and in-context learning tasks, respectively. The selection of these datasets ensures a diverse set of tasks, enabling robust and comprehensive evaluation across the pre-training and fine-tuning stages.

3.1.2 Experimental Details

During pretraining, we employ Llama-3-8B-Instruct as the target LLM and train two types of compressors: a lightweight compressor and a large compressor, which uses the same decoder-only architecture. For the lightweight compressor, we utilize GPT2-Large as the base model, trained using a full-parameter fine-tuning method. The large compressor employs Llama-3-8B-Instruct as its base model, trained with Low-Rank Adaptation (LoRA) (Hu et al., 2021) with a parameter setting of $r = 64$. The learning rates are set to



(a) Converged metrics (Lite) (b) Different compression rate (Lite) (c) Light v.s. Large compressor

Figure 2: The converged metrics and pre-training curves for various configurations in the text reconstruction task.

1×10^{-4} during the pre-training phase. The trainable parameters remain consistent between the pre-training and fine-tuning phases. Pre-training takes 123 hours for PCC (lite) and 204 hours for PCC (large) using four A100 (80GB) GPUs on a node. In all experiments, the decoding method for the LLM decoder is greedy decoding. Further implementation details can be found in Section A.1.

3.1.3 Baselines

We adopt five competitive context compression models as the baselines:

AutoCompressor (Chevalier et al., 2023) We use the princeton-nlp/AutoCompressor-Llama-2-7b-6k checkpoint, which pre-train from Llama-2-7b-hf[3], to generate summary vectors, achieving a compression rate of 40×.

xRAG (Cheng et al., 2024) We use the Hannibal046/xrag-7b checkpoint. It employs Mistral-7B-Instruct-v0.2[4] as its target LLM and is extensively fine-tuned on multiple downstream datasets, including SQuAD, NQ, and TriviaQA.

COCOM (Rau et al., 2024) We adopt the 4x, 16x, and 128x Lite compression models released by the authors, along with the fine-tuned target LLM Mistral-7B-Instruct-v0.2.

ICAE (Ge et al., 2024) We use the sggetao/icae/mistral_7b_ft_icae checkpoint with 4x compression rate. It employs Mistral-7B-Instruct-v0.2 as its target LLM.

LLMLingua-2 (Pan et al., 2024) LLMLingua-2 is an explicit compression method. For our experiments, we use the microsoft/llmlingua-2-xlm-roberta-large-meetingbank checkpoint. The compression rate is set to 4x, and we conduct experiments with Llama-3-8B-Instruct as the target LLM.

3.2 Pre-Training Dynamics

We begin by examining the convergence of pre-training using different configurations.

Text Reconstruction Figure 2a illustrates the ability to reconstruct original text from memory slots. We use commonly-used metrics in machine translation, namely BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004). The results indicate that with compression rates of 4x and 16x, the compressor captures almost all the original content. However, the reconstruction quality drops sharply at 64x compression. Figure 2b presents the training curves, showing that lower compression rates, such as 4x and 16x, quickly converge to a low loss state, while higher compression rates face difficulties in convergence. Switching to a larger compressor only slightly improves the situation but does not change the overall pattern, as shown in Figure 2c. We offer a case study on text reconstruction with PCC in Table A5 in Appendix.

Text Completion Figure 3 presents the training curves for the text completion task. The main conclusions are similar to those in the text reconstruction task, but with two unique characteristics. First, compressors with lower compression rates take longer to converge. For example, the 4x compressor takes the longest time to converge, while compression rates of 64x and above converge quickly but remain at a high loss value. Second, the advantage of larger compressors over smaller ones is much more significant in the text completion task compared to the text reconstruction task. Additionally, we evaluate the perplexity with various number of memory slots in the context, and the results are presented in Table A1 in Appendix A.2.

[3] https://huggingface.co/meta-llama/Llama-2-7b-hf

[4] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

	Dataset	SQuAD		HotPotQA		AdversarialQA		NQ		Average	
	Metrics	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
Reference	w/o Context	14.84	3.41	24.80	14.90	12.05	6.43	27.48	15.02	19.79	9.94
	w/ Context	79.81	59.97	64.60	51.12	56.10	38.67	64.64	51.38	66.29	50.29
Baselines	AutoCompressor	21.46	0.35	16.29	0.29	14.09	2.00	25.57	0.63	19.35	0.82
	xRAG	18.19	3.46	27.51	16.29	13.75	3.47	38.06	20.80	24.38	11.01
	4x COCOM-Lite	21.70	9.17	40.07	32.32	19.45	13.90	50.45	41.87	32.92	24.32
	16x COCOM-Lite	19.23	8.13	31.94	25.27	19.35	14.73	26.36	20.66	24.22	17.20
	128x COCOM-Lite	19.56	7.61	23.63	18.68	19.47	15.13	18.79	14.36	20.36	13.95
	ICAE	45.69	21.63	35.16	26.68	27.98	11.70	59.15	47.35	42.00	26.84
	LLMLingua2	51.20	32.18	**55.72**	**44.18**	35.41	24.80	68.44	55.85	52.69	39.25
PCC (Lite)	4x Memory Context	<u>75.83</u>	<u>57.44</u>	<u>50.37</u>	<u>42.20</u>	<u>50.36</u>	<u>37.83</u>	70.51	<u>61.97</u>	<u>61.77</u>	<u>49.86</u>
	16x Memory Context	56.82	37.72	35.67	27.88	40.94	28.07	62.56	52.73	49.00	36.60
	64x Memory Context	37.89	22.66	20.62	14.27	27.02	16.30	36.82	29.18	30.59	20.60
	128x Memory Context	32.24	15.92	22.98	16.92	25.18	15.73	44.25	36.08	31.16	21.16
	256x Memory Context	26.71	12.63	18.69	12.92	22.32	13.93	35.69	27.86	25.85	16.84
PCC (Large)	4x Memory Context	**77.76**	**60.04**	48.19	39.97	**52.56**	**39.37**	**75.96**	**67.71**	**63.62**	**51.77**
	16x Memory Context	55.32	37.72	33.76	25.27	38.96	26.47	<u>71.09</u>	<u>61.97</u>	49.78	37.86
	128x Memory Context	33.72	18.86	27.60	20.44	29.75	21.13	62.57	53.30	38.41	28.43

Table 2: Performance comparison of various methods on the QA task across four datasets. NQ stands for Natural Questions dataset, EM indicates Exact Match. The **best-performing results** on each dataset are highlighted in bold, while the <u>second-best results</u> are underlined. To address potential inconsistencies in decoder configurations across baselines, we further evaluate our method using alternative decoders to ensure fairness. We provide additional detailed results in Table A2 in the Appendix.

The results show that our model can generalize to multiple segments well.

3.3 Results of Downstream Tasks

3.3.1 RAG-based QA

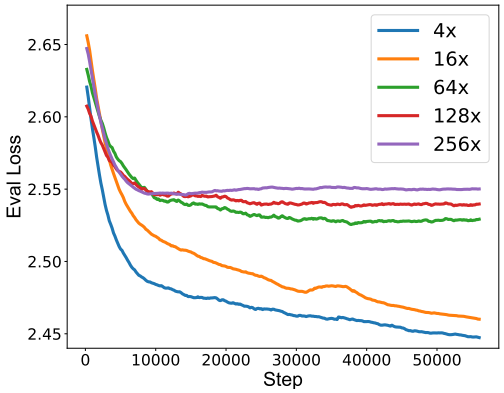
The first type of downstream task is retrieval-augmented generative question-answering (RAG-based QA). Each data sample consists of a triple: (context, question, answer). *Reference (w/o Context)* means prompting the target LLM only with the question, while *Reference (w/ Context)* means concatenating the context and question as the prompt. For all compression models, including both our PCC and baselines, they compress the context and then concatenate the compressed memory with the original question as the prompt. To demonstrate the generalization ability of our model, for PCC models, we fine-tune only using the training set of SQuAD and then evaluate on SQuAD, HotPotQA (Yang et al., 2018), AdversarialQA (Bartolo et al., 2020), and NQ (Karpukhin et al., 2020). For baselines, we use their released settings, so they may be fine-tuned with multiple training sets, such as xRAG.

We evaluate performance using F1 and Exact Match (EM) metrics, where F1 score denotes the harmonic mean of precision and recall, and EM evaluates whether the predicted response precisely aligns with the ground truth answer.

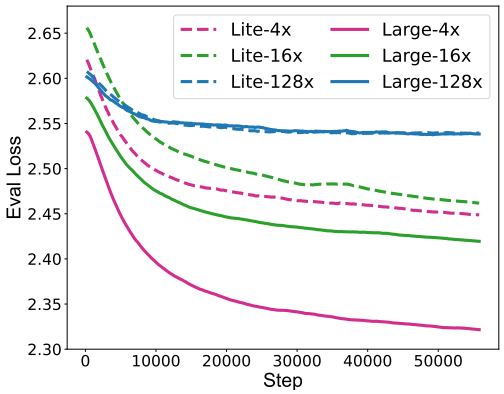
Table 2 reports the overall results. First, in most cases and in terms of average scores, PCC (large) performs best, followed by PCC (lite). This verifies the efficacy of our method. The only exception is on HotPotQA, where LLMLingua2 outperforms our models. However, our PCC (Large) 4x outperforms LLMLingua2 in all the other three datasets. Second, 4x compression preserves the information from the context well for both lite and large compressors, sacrificing only slightly in performance compared to the reference model with the full context as explicit input. With larger compression rates, the performance drops more significantly. However, even with a compression rate as high as 256x, PCC still outperforms Reference (w/ Context), indicating that the memory still captures useful information in the highly compressed memory slots. Furthermore, our models demonstrate strong generalization ability, performing well on the HotPotQA, AdversarialQA, and NQ datasets without fine-tuning on their training sets.

3.3.2 In-Context Learning

The challenge of RAG-based QA lies in learning to locate useful evidence from the augmenting context for answer generation. In contrast, the in-context learning (ICL) task requires the ability to infer latent patterns from demonstrations in the augmenting context and apply these patterns to the current task. This task is more challenging than direct evidence identification and thus warrants a separate



(a) Comparison across different compression rates (Lite)



(b) Comparison between Lite and Large compressors

Figure 3: The pre-training curves with various configurations in the text completion task.

Methods	GSM8K	SST-2	WSC
Zero-shot	64.82	87.04	53.85
ICL 150 Tokens	71.72	93.92	50.00
ICL 750 Tokens	**78.92**	94.19	45.83
4x Comp. 750 Tokens	74.91	**94.61**	**69.55**
4x Comp. 1500 Tokens	63.76	93.50	64.42
16x Comp. 750 Tokens	71.65	93.16	62.18
16x Comp. 1500 Tokens	67.48	89.99	61.54

Table 3: Comparison of performance in three ICL datasets. Accuracy (%) is employed as the metrics. Bold values indicate the highest and underlined values denote the second-highest accuracy. The compressor employed here is PCC (Large).

Category	Methods	PPL
Reference	Zero-shot	36.45
	750 tokens context	27.74
	1500 tokens context	26.08
SFT	Zero-shot	24.39
	750 tokens context	**17.38**
	1500 tokens context	19.18
PCC (Lite)	4x 750 tokens context	19.57
	4x 1500 tokens context	20.31
	16x 750 tokens context	32.46
	16x 1500 tokens context	30.49

Table 4: Perplexity (PPL) in the role-play task. Lower PPL indicates better performance in generating the target character's responses.

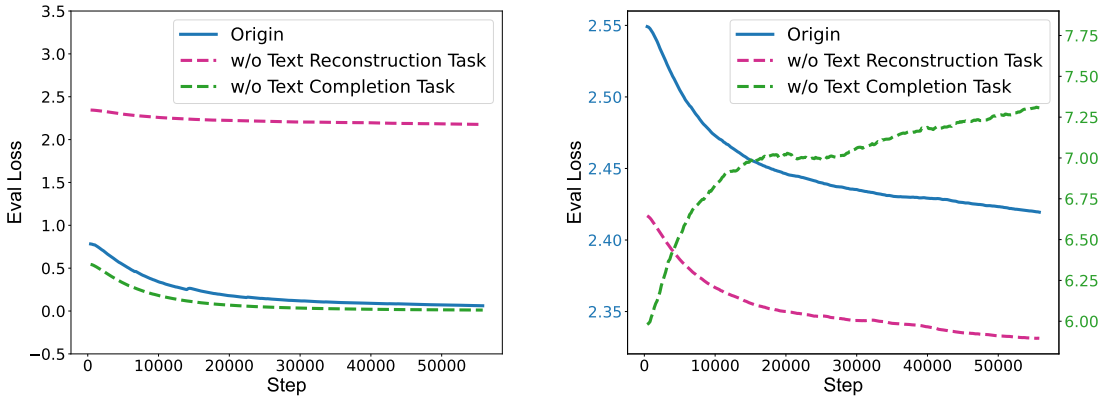
testing scenario. We use three datasets for evaluation: GSM8K (mathematical reasoning), SST-2 (sentiment analysis) (Socher et al., 2013), and WSC (Winograd Schema Challenge) (Levesque et al., 2012). PCC (Large) is fine-tuned with the training set of GSM8K. We compare zero-shot generation, few-shot ICL with a maximum context of 150 tokens and 750 tokens, and PCC (Large) with 4x and 16x compressed in-context examples. Results are reported in Table 3. PCC performs very well in the ICL scenario, with a 4x compression rate performing on par with or even better than explicit ICL with 750 tokens.

3.3.3 Role-Playing

Role-playing is a critical capability of LLMs that facilitates many applications, such as social simulations, AI companions, and digital twins. A commonly used strategy for LLM-based role-playing tasks is to include the character's profile and past experiences (such as historical utterances and behaviors) in the prompt. This makes the task an ideal playground for memory representation. To evaluate the effectiveness of context compression in the role-playing scenario, we use the Harry Potter Dialogue dataset. PCC (Lite) is fine-tuned on a hold-out training set and trained with the compressed 6-shot examples, while SFT (Supervised Fine-Tuning) is fine-tuned on the same number of examples but without compressing the 6-shot examples. A lower perplexity (PPL) indicates better capture of the character's target utterance by the model. The results, presented in Table 4, shows that PCC demonstrates excellent efficacy in this scenario. With a 4x compression rate, it outperforms the explicit prompt-based role-playing strategy, with only a slight performance degradation compared to the SFT method. At 16x compression, its performance lies between zero-shot role-playing and few-shot role-playing.

3.4 Connecting Various LLMs

To assess whether the pre-trained compressor can enhance the performance of various LLMs, we con-



(a) Training loss in the text reconstruction task

(b) Training loss in the text completion task

Figure 4: Ablation study: pre-training curves for PCC (Large) with 16x compression rate when some components are removed during the pre-training stage. And *Origin* represents original LLM.

Model	Methods	F1	EM
Mistral-7B	w/o Context	11.33	1.25
	w Context	49.02	23.89
	4x Comp.	46.57	35.15
	16x Comp.	35.29	20.55
Qwen2.5-7B	w/o Context	25.91	16.48
	w Context	66.51	49.95
	4x Comp.	46.39	32.77
	16x Comp.	31.52	16.90
Phi-3.5-mini	w/o Context	11.45	2.24
	w Context	60.53	40.66
	4x Comp.	48.01	34.97
	16x Comp.	39.30	27.30
Llama-3-8B	w/o Context	19.79	9.94
	w Context	66.29	50.29
	4x Comp.	61.77	49.86
	16x Comp.	49.00	36.60

Table 5: Performance of four different downstream LLMs. The metrics employed are the average F1 score and Exact Match (EM) on four datasets: SQuAD, HotPotQA, AdversarialQA, and Natural Questions (NQ). The compressor employed here is PCC (Lite).

duct experiments using three additional LLMs as downstream decoders: Mistral-7B-Instruct-v0.2, Qwen2.5-7B-Instruct, and Phi-3.5-mini-instruct, within the RAG-based QA domain. For each decoder, using the pre-training checkpoint described in Section 2.3, we perform lightweight fine-tuning on the compressor with 32 million pre-training tokens and the SQuAD training set to ensure alignment with the frozen decoder. The results are presented in Table 5 (for the complete table, please refer to Table A2 in the Appendix). Across all downstream LLMs, the pre-trained compressor (PCC) demonstrates its effectiveness by providing valuable context from the compressed memory slots, showing significant improvement compared to the models' performance without context.

Methods	F1	EM
Reference (w/o Context)	19.79	9.94
PCC (Large) 16x	**49.78**	**37.86**
w/o Pretrain Stage	23.51	15.96
w/o Text Reconstruction Task	47.65	36.74
w/o Text Completion Task	46.79	35.65
w/o Mutil Special Tokens	47.87	36.36

Table 6: Ablation study: Average performance of four datasets on the RAG-based QA domain. The metrics reported are in percentages (%).

3.5 Ablation Study

In addition to the compression rate and compressor model size, which have been extensively studied in previous experiments, other important components of PCC include the pre-training stage, specific pre-training tasks, and the memory slots wrapper (<MEM> </MEM>). To further investigate these components, we conduct an ablation study. For example, *w/o PretrainStage* indicates removing the entire pre-training stage and retaining only the fine-tuning stage with the SQuAD training set. Results are reported in Table 6 (for the full table, please refer to Table A3 in the Appendix). We find

that the pre-training stage is particularly crucial for generating high-quality memory slots. Removing any component from the PCC framework results in a noticeable performance drop, highlighting the importance of this comprehensive approach to pre-training and memory representation. We further demonstrate the changes in the pre-training loss curves when either the text reconstruction or the text completion tasks are removed in Figure 4. It is interesting to observe that when only the text completion task is used in pre-training (as shown in Figure 4a), the text reconstruction ability degrades significantly. Conversely, when only the text reconstruction task is used in pre-training (as shown in Figure 4b), the text completion task fails to converge. The synergy of these two tasks makes the pre-training of the compressor more robust.

4 Related Work

LLMs with Long Context. Perceiving long context in large language models (LLMs) has become ubiquitous with numerous real-world applications, such as RAG systems (Gao et al., 2023), information extraction from social platforms (Shang et al., 2024), agentic frameworks for task automation (Wang et al., 2024; Huang et al., 2023), and role-playing agents (Zhong et al., 2024; Liu et al., 2024), where LLMs need to remember extensive character profiles, experiences, and past conversations. Pretraining LLMs to be context-aware over long sequences is expensive, so researchers often develop context extension techniques to expand the context windows of well-pretrained LLMs (Chen et al., 2023b; Ding et al., 2024; Zhang et al., 2024b). Given the high computational and storage costs associated with long-sequence inference, researchers have explored various methods to reduce these expenses (Zhang et al., 2024a; Jiang et al., 2024; Fang et al., 2024; Munkhdalai et al., 2024). However, these methods typically require modifications or fine-tuning of the LLMs. In this paper, we examine methods that compress long contexts into shorter versions while keeping the LLMs frozen. Thus, our focus is on context compression.

Context Compression. There are primarily two approaches for context compression: explicit compression and implicit compression. In the explicit compression approach, researchers develop methods to identify and remove redundant and non-essential tokens in the context, retaining only the important and compact information(Li et al., 2023; Jiang et al., 2023; Pan et al., 2024). This approach has the advantage of explainability, as it is possible to see which tokens are removed from the original context. However, the utilization of context can be suboptimal since it is limited to explicit tokens. In the implicit compression approach, researchers compress the original context into implicit memory slots, leveraging the fact that embedding representations can effectively condense key information (Chevalier et al., 2023; Cheng et al., 2024; Ge et al., 2024; Li et al., 2024b; Rau et al., 2024). This paper aims to push the frontier of the second approach forward by addressing key questions in context compression from a systematic perspective, including pretraining, universal and decoupled module design, compressor size and rate, and efficacy across different types of downstream applications.

5 Conclusion

In this paper, we focus on efficient long contexts processing for LLMs using embedding-based compression. We propose a decoupled compressor-LLM framework and advocate for pre-training with two language modeling tasks. The resulting method, PCC, serves as a universal and lightweight compressor adaptable to various downstream LLM decoders. Through rigorous experiments, we demonstrate that compression rates of 4x and 16x strike a good balance between accuracy and efficiency, while higher rates like 256x capture useful context at the expense of some information loss. Notably, our method outperforms competitive context compression baselines across diverse downstream tasks. We hope this research highlights the potential of implicit context compression to enhance LLM efficiency and lays the groundwork for future innovations in scalable LLM inference.

6 Limitation

While our work demonstrates promising results in embedding-based compression for LLMs, we acknowledge several limitations that provide opportunities for future improvement.

- Investigating different configurations of compressor pre-training is both time and GPU consuming. For the current version, we can only afford up to 8B LLM's pretraining experiments. While this paper advocates for a lightweight compressor, we hope in the next version, we can launch larger compressors such as 70B models to investigate

the upper bounds of implicit compression. This may break the dilemma of higher compression rates, like 256x, being highly efficient but losing too much useful information.

- The current design of PCC compresses segments of a long context into even memory slots. However, considering that different segments of text may convey different levels of information mass and redundancy, an even compression is not an optimal approach. In the future, we plan to investigate a dynamic implicit compression technique that can determine the compression rate for each segment adaptively.

- Although currently the adaptation of the compressor to various downstream LLMs only requires a few steps of fine-tuning, it still necessitates parameter updates on both the compressor encoder and compressor converter to achieve desirable performance. In the future, we hope to develop a solution where only the converter needs to be fine-tuned, allowing the compressor encoder to remain frozen, thereby making the adaptation process even more lightweight.

7 Acknowledgments

The authors from SZU acknowledge the financial support from the National Natural Science Foundation of China (Grant Nos. 62276171, 62476173, 62002233, 61972145), the Shenzhen Fundamental Research-General Project (Grant Nos. JCYJ20240813142610014, JCYJ20240813141503005, JCYJ20220811155801-3001), Guangdong Basic and Applied Basic Research Foundation (Grant Nos. 2024A1515011938 and 2020B1515120028), Guangdong Peral River Recruitment Program of Talents (2019ZT08X603). Hao Liao is the corresponding author.

References

Max Bartolo, Alastair Roberts, Johannes Welbl, Sebastian Riedel, and Pontus Stenetorp. 2020. Beat the ai: Investigating adversarial human annotation for reading comprehension. *Transactions of the Association for Computational Linguistics*, 8:662–678.

Nuo Chen, Yan Wang, Haiyun Jiang, Deng Cai, Yuhan Li, Ziyang Chen, Longyue Wang, and Jia Li. 2023a. Large language models meet harry potter: A dataset for aligning dialogue agents with characters. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8506–8520.

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

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. 2024. xrag: Extreme context compression for retrieval-augmented generation with one token. *arXiv preprint arXiv:2405.13792*.

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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 3829–3846. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Junjie Fang, Likai Tang, Hongzhe Bi, Yujia Qin, Si Sun, Zhenyu Li, Haolun Li, Yongjian Li, Xin Cong, Yankai Lin, et al. 2024. Unimem: Towards a unified view of long-context large language models. *arXiv preprint arXiv:2402.03009*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

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*.

Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. 2023. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505*.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. 2024. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 13358–13376. Association for Computational Linguistics.

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*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Thirteenth international conference on the principles of knowledge representation and reasoning*.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024a. Long-context llms struggle with long in-context learning. *Preprint*, arXiv:2404.02060.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6342–6353. Association for Computational Linguistics.

Zongqian Li, Yixuan Su, and Nigel Collier. 2024b. 500xcompressor: Generalized prompt compression for large language models. *arXiv preprint arXiv:2408.03094*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Jiongnan Liu, Yutao Zhu, Shuting Wang, Xiaochi Wei, Erxue Min, Yu Lu, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. 2024. Llms+ persona-plug= personalized llms. *arXiv preprint arXiv:2409.11901*.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 963–981. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

David Rau, Shuai Wang, Hervé Déjean, and Stéphane Clinchant. 2024. Context embeddings for efficient answer generation in rag. *arXiv preprint arXiv:2407.09252*.

Tianqi Shang, Weiqing He, Tianlong Chen, Ying Ding, Huanmei Wu, Kaixiong Zhou, and Li Shen. 2024. Integrating social determinants of health into knowledge graphs: Evaluating prediction bias and fairness in healthcare. *Preprint*, arXiv:2412.00245.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

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*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2024a. Long context compression with activation beacon. *arXiv preprint arXiv:2401.03462*.

Peitian Zhang, Ninglu Shao, Zheng Liu, Shitao Xiao, Hongjin Qian, Qiwei Ye, and Zhicheng Dou. 2024b. Extending llama-3's context ten-fold overnight. *arXiv preprint arXiv:2404.19553*.

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.

A Appendix

A.1 Pipline Implementation Details

Table A4 presents the hyperparameters utilized for training both the lightweight and large compressor models. Both models are optimized using the AdamW algorithm in conjunction with a cosine learning rate scheduler. The learning rate is adapted for different training phases, including pretraining and fine-tuning on the SQuAD and GSM8K datasets. Batch sizes and warmup steps are specifically configured for each model and task, with a maximum gradient norm of 0.5 consistently applied. For the lightweight compressor, all parameters are trained directly, whereas the large compressor employs LoRA with a rank of 64, a scaling factor of $\alpha = 32$, and a dropout rate of 0.1. Training is conducted on 4xA100 and 8xH100 GPUs, each equipped with 80 GB of memory, ensuring efficient computation for both model types. The lightweight compressor completes one epoch during pretraining and three epochs during fine-tuning, while the large compressor adheres to the same schedule for fine-tuning tasks.

A.2 Perplexity Evaluation

The effect of memory embedding on the perplexity of the target LLM is evaluated by calculating the perplexity of the second half of the input tokens. In the Plain method, as described in Reference, the first half of the input tokens is discarded, whereas the Context method retains the complete input sequence. Conversely, the PCC method compresses the first half of the input tokens into memory embeddings. Detaled experimental results are provided in Table A1.

A.3 Inference Efficiency

We evaluate the enhancement in inference efficiency of the target LLM using the lightweight compressor across various input combinations. Given that our method allows for the pre-caching of context memory embeddings, we also investigat the impact of caching these embeddings on the improvement of inference efficiency. The experimental results are presented in Table A7.

The machine configuration used for testing latency. The hardware setup included an AMD EPYC 7V13 64-Core Processor for the CPU, an A100 80GB PCIe GPU for computation, and 216GB of RAM for memory-intensive operations. The version of software are Python 3.11, PyTorch 2.7.0, Transformers 4.41.2 and CUDA 12.6.

A.4 Case Study

The performance of the model on the text reconstruction task is evaluated by compressing a 256-token text sequence with compression rates of 4x, 16x, and 64x, respectively. The original text and the reconstructed text are presented in Table A5. In the 4x and 16x compression scenarios, the reconstructed text closely aligned with the original, achieving BLEU scores of 100% and 98.8%, respectively, demonstrating the model's high fidelity in text restoration at low and moderate compression rates. However, as the compression rate increased to 64x, performance deteriorated significantly, with the BLEU score dropping to 27.8%.

Samples from the NQ dataset are selected for testing to evaluate performance. Unlike COCOM, which is fine-tuned on the target LLM, our proposed method is not fine-tuned on the target model. Despite this, our method outperforms COCOM in the evaluation. However, it can be observed that as the compression rate increases, the ability of the memory embedding to effectively extract and retain useful information gradually diminish.

	Input Tokens	512	1024	2048	4096
Reference	w/o Context	17.31	14.70	13.07	11.99
	w/ Context	12.38	11.57	11.00	10.57
PCC (Lite)	4x Comp.	11.22	10.97	10.82	11.30
	16x Comp.	12.14	11.78	11.48	11.78
	64x Comp.	13.05	12.46	12.42	13.02
	128x Comp.	13.13	12.42	11.88	11.57
	256x Comp.	13.24	13.15	12.69	12.13
PCC (**Large**)	4x Comp.	**10.66**	**10.82**	**10.79**	**11.10**
	16x Comp.	11.84	11.64	11.28	11.23
	128x Comp.	13.01	12.51	12.26	12.28

Table A1: Perplexity (PPL) trends across different context lengths and memory slot configurations for the pre-training text completion task. Each context length is evenly divided, with the first half used to generate memory slots and the second half used for PPL computation.

	Dataset	SQuAD		HotPotQA		AdversialQA		NQ		Average	
	Metrics	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
Mistral	w/o Context	10.34	0.69	8.44	0.13	7.80	0.83	18.75	3.36	11.33	1.25
	w Context	56.30	27.33	52.32	30.97	36.96	12.60	50.51	24.64	49.02	23.89
	4x Comp.	62.40	44.12	31.97	24.52	42.51	30.70	49.41	41.28	46.57	35.15
	16x Comp.	45.89	25.61	23.91	17.92	30.23	19.02	41.12	19.63	35.29	20.55
Qwen	w/o Context	21.03	7.09	30.66	23.29	16.55	10.53	35.41	25.00	25.91	16.48
	w Context	78.30	57.44	63.30	51.73	55.10	36.17	69.33	54.47	66.51	49.95
	4x Comp.	56.79	36.16	38.09	28.84	37.55	25.70	53.14	40.40	46.39	32.77
	16x Comp.	34.01	14.53	30.98	20.64	25.01	12.97	36.06	19.46	31.52	16.90
Phi	w/o Context	11.49	1.04	14.11	4.14	8.64	1.70	11.55	2.08	11.45	2.24
	w Context	73.12	46.71	59.03	44.31	50.22	29.27	59.75	42.34	60.53	40.66
	4x Comp.	57.63	36.68	36.43	27.52	41.63	29.20	56.33	46.50	48.01	34.97
	16x Comp.	44.29	26.47	32.81	24.74	32.14	20.33	47.94	37.65	39.30	27.30

Table A2: Performance evaluation of three LLMs—Mistral 7B v0.2 Instruct, Qwen-2.5-instruct, and Phi-3.5-mini-instruct—on four widely-used question-answering datasets: SQuAD, HotPotQA, AdversarialQA, and NQ. F1 score (%) and Exact Match (EM, %) are utilized as evaluation metrics. Each model is assessed under three conditions: (1) *Without Context*, where no additional context is provided during evaluation; (2) *With Context*, where relevant context is included; and (3) *4x/16x Compression*, which evaluates model performance after applying a 4x/16x compressor. The average scores across all datasets are presented in the last columns for a comprehensive comparison.

	Dataset	SQuAD		HotPotQA		AdversarialQA		NQ		Average	
	Metrics	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
Reference	w/o Context	14.84	3.41	24.80	14.90	12.05	6.43	27.48	15.02	19.79	9.94
	w/ Context	79.81	59.97	64.60	51.12	56.10	38.67	64.64	51.38	66.29	50.29
Baselines	AutoCompressor	21.46	0.35	16.29	0.29	14.09	2.00	25.57	0.63	19.35	0.82
	xRAG	18.19	3.46	27.51	16.29	13.75	3.47	38.06	20.80	24.38	11.01
	4x COCOM-Lite	21.70	9.17	40.07	32.32	19.45	13.90	50.45	41.87	32.92	24.32
	16x COCOM-Lite	19.23	8.13	31.94	25.27	19.35	14.73	26.36	20.66	24.22	17.20
	128x COCOM-Lite	19.56	7.61	23.63	18.68	19.47	15.13	18.79	14.36	20.36	13.95
	ICAE	45.69	21.63	35.16	26.68	27.98	11.70	59.15	47.35	42.00	26.84
	LLMLingua2	51.20	32.18	**55.72**	**44.18**	35.41	24.80	68.44	55.85	52.69	39.25
Ablation	16x Memory Context	**55.32**	**37.72**	33.76	25.27	**38.96**	26.47	**71.09**	61.97	**49.78**	**37.86**
	w/o Pretrain Stage	19.80	8.13	25.21	19.10	15.93	11.00	33.10	25.61	23.51	15.96
	w/o Text Completion Task	47.76	29.24	33.54	25.72	35.71	25.33	70.15	62.29	46.79	35.65
	w/o Text Reconstruction Task	44.86	28.55	36.34	28.39	38.53	**27.53**	70.88	**62.47**	47.65	36.74
	w/o Mutil Special Tokens	51.92	33.39	31.25	23.71	38.27	27.17	70.05	61.18	47.87	36.36

Table A3: Detailed results of the 16x compression configuration using the large compressor in the ablation study for QA tasks. The table presents the performance (F1 and Exact Match (EM)) across four datasets: SQuAD, HotPotQA, AdversarialQA, and NQ, along with the average metrics. Methods compared include baseline models, ablation variations (e.g., removal of pretraining, text completion,text reconstruction tasks, and multi-special tokens), and the full model configurations. Best-performing results for each dataset are highlighted in bold, and second-best results are underlined.

Hyperparameter	**PCC (Lite)**	**PCC (Large)**
Optimizer	AdamW	AdamW
Learning rate	1e-4 (Pre-train) 5e-5 (SQuAD Fine-tune) 1e-5 (HPD Fine-tune)	1e-4 (Pre-train) 5e-5 (SQuAD Fine-tune) 1e-5 (GSM8K Fine-tune)
Lr schedular	Cosine	Cosine
Batch size	256 (Pre-train) 128 (SQuAD Fine-tune) 8 (HPD Fine-tune)	256 (Pre-train) 128 (SQuAD Fine-tune) 32 (GSM8K Fine-tune)
Warmup steps	300 (Pre-train) 100 (Fine-tune)	300 (Pre-train) 100 (Fine-tune)
Max grad norm	0.5	0.5
Epochs	1 (Pre-train) 3 (Fine-tune)	1 (Pre-train) 3 (Fine-tune)
Training method	Full parameters	LoRA ($r = 64$, $\alpha = 32$, dropout=0.1)
Pre-training Time	123 Hours(4 $\times$ A100 80GB) –	204 Hours(4 $\times$ A100 80GB) 67 Hours(8 $\times$ H100 80GB)

Table A4: Hyperparameters for training the light-weight and large compressor models.

Origin the numbers used in the formula on your IRS Form 1040. Here's where each input you'll need is located: -Total tax: Line 24 -Taxable income: Line 15 Worth noting, income refers to both earned and unearned income: -Earned income: Wages, commissions, salary, and bonuses -Unearned income: Interest income on saving accounts, dividends from stocks, and bond interest As an example, consider an individual who paid $24,000 in total tax and had a taxable income of $95,000 in salary, a $5,000 bonus, and $1,000 in income from savings interest and dividends. This person's total taxable income would be $95,000 + $5,000 + $1,000 = $101,000. Individual effective tax rate = 24,000 / 101,000 = 0.2376 You can then express this as a percentage by multiplying it by 100 to arrive at 23.76%.	**4x** the numbers used in the formula on your IRS Form 1040. Here's where each input you'll need is located: -Total tax: Line 24 -Taxable income: Line 15 Worth noting, income refers to both earned and unearned income: -Earned income: Wages, commissions, salary, and bonuses -Unearned income: Interest income on saving accounts, dividends from stocks, and bond interest As an example, consider an individual who paid $24,000 in total tax and had a taxable income of $95,000 in salary, a $5,000 bonus, and $1,000 in income from savings interest and dividends. This person's total taxable income would be $95,000 + $5,000 + $1,000 = $101,000. Individual effective tax rate = 24,000 / 101,000 = 0.2376 You can then express this as a percentage by multiplying it by 100 to arrive at 23.76%. (Exactly recover.)
16x the numbers used in the formula on your IRS Form 1040. Here's where each input you'll need is located: -Total tax: Line 24 -Taxable income: Line 15 Worth noting, income refers to both earned and unearned income: -Earned income: Wages, commissions, salary, and bonuses -Unearned income: Interest income on saving accounts, dividends from stocks, and bond interest As an example, consider an individual who paid $24,000 in total tax and had a taxable income of $95,000 in salary, a $5,000 bonus, and $1,000 in income from savings interest and dividends. This person's total taxable income would be $95,000 + $5,000 + $1,000 = $101,000 Individual effective tax rate = 24,000 / 101,000 = 0.2376 You can then express this as a percentage by multiplying it by 100 to arrive at 23.76%. (Almost Exactly recover.)	**64x** the numbers you'll use on the IRS Form 1040. Here's where each hlline is located: - Total income: Taxable income. - Worthy of note: Taxable income includes both earned and unearned income. Taxable income includes: - Wages: Earned income from salaries, commissions, and bonuses on an hourly basis. - Unearned income: Interest, dividends, and capital gains from saving and investing in stocks, bonds, and a total return account. As an example, consider a person who had $25,000 in taxable income and paid $5,000 in taxes on an income of $100,000 in salary and interest. Total person's taxable income would be $100,000 + $25,000 = $125,000. This person's effective tax rate would be 100,000 / 125,000 = 80%. Individuals can then express this as a percentage by multiplying 80% by 100 to get 80. Effective tax rate = 80%

Table A5: Case study: Reconstructed text using different compression rates (4, 16, and 64). The compressor employed here is PCC (Lite). And *Origin* represents original LLM. Dataset: FineWeb. Highlighted words indicate inconsistencies between the restored text and the original text.

Model input
Question: When did the hawks win the NBA championship? **Passage 1**: Atlanta Hawks the NBA as part of the merger between the NBL and the Basketball Association of America (BAA), and briefly had Red Auerbach as coach. In 1951, Kerner moved the team to Milwaukee, where they changed their name to the Hawks. Kerner and the team moved again in 1955 to St. Louis, where they won their only NBA Championship in 1958 and qualified to play in the NBA Finals in 1957, 1960 and 1961. The Hawks played the Boston Celtics in all four of their trips to the NBA Finals. The St. Louis Hawks moved to Atlanta in 1968, when Kerner **Passage 2**: 1958 NBA Finals The 1958 NBA World Championship Series was the championship series for the 1957–58 National Basketball Association (NBA) season, and the conclusion of the seasonś playoffs. It pitted the Western Division champion St. Louis Hawks against the Eastern Division champion Boston Celtics. The Hawks won the series in six games to win the clubś first and so far only NBA championship title. "Hawks win series 4–2" After suffering a heartbreaking loss to the Celtics in Game 7 of the 1957 NBA Finals, St. Louis survived a sometimes difficult 1957-58 NBA season, returning to the NBA Finals to face **Passage 3**: 1971 NBA Finals The 1971 NBA World Championship Series was the championship series played at the conclusion of the National Basketball Association (NBA)'s 25th anniversary season of 1970–71. The Western Conference champion Milwaukee Bucks, who were founded just three years earlier, swept the Eastern Conference champion Baltimore Bullets in four games. Baltimore had dethroned the 1969–70 NBA champion New York Knicks. The Bucks were the first Western Conference champions to win the league's finals since the St. Louis Hawks did so in 1958. This was the first NBA Finals not played in the state of California in 10 years. It
Generated Response
Label: 1958 **LLM w/o passages**: The Atlanta Hawks have never won an NBA championship. **LLM w/ passages**: 1958 **PCC Lite 4x Comp**: 1958 **PCC Lite 16x Comp**: 1958 **PCC Lite 128x Comp**: 1958 **PCC Lite 256x Comp**: 1957 **COCOM 4x Comp**: 1958 **COCOM 16x Comp**: 2006 **COCOM 128x Comp**: The Nba is a professional wrestling championship

Table A6: Case Study: Generated responses using different methods. The LLM used is Llama-3-8B-Instruct, and the compressors applied are the lightweight versions of PCC and COCOM. Dataset: NQ.

Batch×Input	Model	Compression Time	Prefilling Time	Decoding Time	Total Time
16×1024	Origin	-	1378.79	1437.60	2816.39
	4x Comp.	604.71	359.20	956.97	1920.88
	16x Comp.	509.83	100.95	1090.76	1701.54
	128x Comp.	500.88	35.87	1051.63	1588.38
32×1024	Origin	-	2774.99	2372.02	5147.01
	4x Comp.	1189.60	709.19	1087.64	2986.43
	16x Comp.	998.91	188.27	1047.38	2234.56
	128x Comp.	975.19	37.49	1057.84	2070.52
16×2048	Origin	-	2830.40	2311.84	5142.24
	4x Comp.	1217.97	718.31	1037.90	2974.18
	16x Comp.	1025.82	188.61	1069.69	2284.12
	128x Comp.	1000.76	37.30	1059.60	2097.66
8×4096	Origin	-	2910.01	2320.38	5230.39
	4x Comp.	1257.87	720.62	1062.48	3040.97
	16x Comp.	1076.08	189.72	1068.69	2334.49
	128x Comp.	1047.36	37.13	1082.19	2166.68
4×8192	Origin	-	3091.50	2145.15	5236.65
	4x Comp.	1343.40	730.81	1084.96	3159.17
	16x Comp.	1145.68	187.20	1032.63	2365.51
	128x Comp.	1114.67	36.75	1058.20	2209.62

Table A7: Inference speed test experiment for lightweight compressor. The unit used in the table is milliseconds. The decoding time is the time it takes to generate 32 tokens. The value on the left side of the bracket represents the acceleration factor of uncached memory embedding, and the value on the right side represents the acceleration factor of cached memory embedding. And *Origin* represents the original LLM.

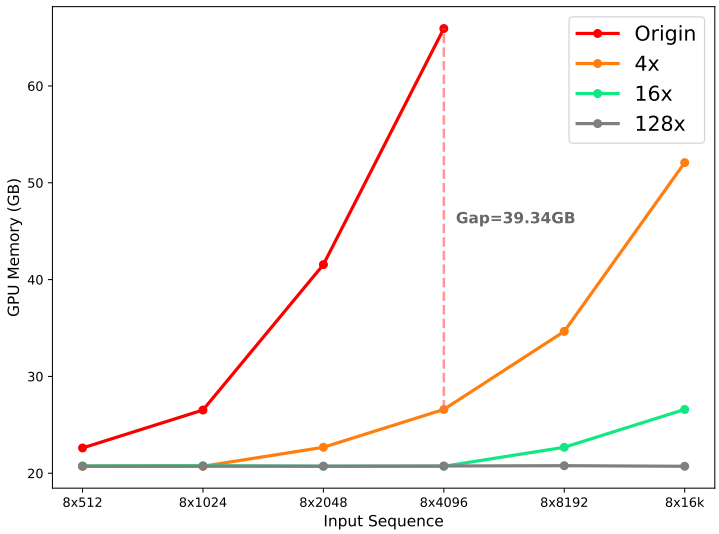


Figure A1: We compare the memory usage during inference between the original LLM method and the PCC method. The original LLM method, represented by *Origin*, runs out of memory (OOM) when the input text has a batch size of 8 and a length of 8192 tokens, exceeding 80 GB of GPU memory. In contrast, the PCC method maintains a much lower memory footprint under the same conditions.