A Silver Bullet or a Compromise for Full Attention? A Comprehensive Study of Gist Token-based Context Compression

Chenlong Deng[1†], **Zhisong Zhang**[3*], **Kelong Mao**[1,2], **Shuaiyi Li**[3], **Xinting Huang**[3], **Dong Yu**[3], **Zhicheng Dou**[1*]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]Beijing Key Laboratory of Research on Large Models and Intelligent Governance
[3]Tencent AI Lab
{dengchenlong,dou}@ruc.edu.cn
zhisonzhang@tencent.com

Abstract

In this work, we provide an empirical investigation of gist-based context compression methods to improve context processing in large language models. We focus on two key questions: (1) How well can these methods replace full attention models? and (2) What potential failure patterns arise due to compression? Through extensive experiments, we show that while gist-based compression can achieve only slight performance loss on tasks like retrieval-augmented generation and long-document QA, it faces challenges in tasks like synthetic recall. Furthermore, we identify three key failure patterns: lost by the boundary, lost if surprise, and lost along the way. To mitigate these issues, we propose two effective strategies: fine-grained autoencoding, which enhances the reconstruction of original token information, and segment-wise token importance estimation, which adjusts optimization based on token dependencies. Our work provides valuable insights into the understanding of gist token-based context compression and offers practical strategies for improving compression capabilities.

1 Introduction

Large language models (LLMs) are increasingly recognized as a key pathway toward general artificial intelligence (OpenAI, 2023; Zhao et al., 2023), with long-context processing emerging as a critical research frontier (Chen et al., 2023; Peng et al., 2024). This capability is crucial for advanced applications like retrieval-augmented generation (RAG), long-term memory systems, and complex reasoning frameworks (Gao et al., 2023; Zhu et al., 2023; Zhang et al., 2024c; Wei et al., 2022; Lightman et al., 2024). Despite the proliferation of architectural innovations, Transformer-based models remain the performance standard. However, these architectures face significant computational challenges when processing extended text sequences: the key-value (KV) cache memory grows linearly with sequence lengths, while the attention mechanism's quadratic calculation introduces substantial overhead. In models like Llama3-8B (Meta-Llama, 2024), a 128K context KV cache can consume memory equivalent to the entire model's parameters, limiting deployment on edge devices and constraining context windows.

A promising approach to mitigate these challenges involves reducing overhead by compressing the number of past tokens stored in the KV cache. This work focuses on a specific type of compression method that condenses the context into a small set of special tokens, called gist tokens (Mu et al., 2023).[1] By replacing the original tokens with a limited number of gist tokens, these methods effectively reduce both KV cache size and computational cost. While such techniques have been successfully applied in real-world tasks (Qian et al., 2024), two critical questions remain unresolved:

Q1: To what extent can this architecture replace full attention models? *Q2*: Does the compression introduce potential, yet significant, failure patterns?

In this work, we thoroughly investigate these two questions through extensive experiments. Specifically, we propose a unified framework for categorizing existing gist-based model architectures along two dimensions: *Memory Location* and *Gist Granularity*. We provide comprehensive evaluations for them with a wide range of language tasks.

For *Q1*, our findings indicate that the fine-grained KV cache architecture (referred to as *Fine KV*) is highly effective, achieving only slight performance loss on various tasks, such as RAG, long-document QA, when compared to the full attention model. However, it still exhibits notable gaps in

[†]This work was done during internship at Tencent AI Lab.
[*]Corresponding authors.

[1]For consistency, we unify the different names for this concept in prior work as "gist tokens" in this paper.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4861–4879

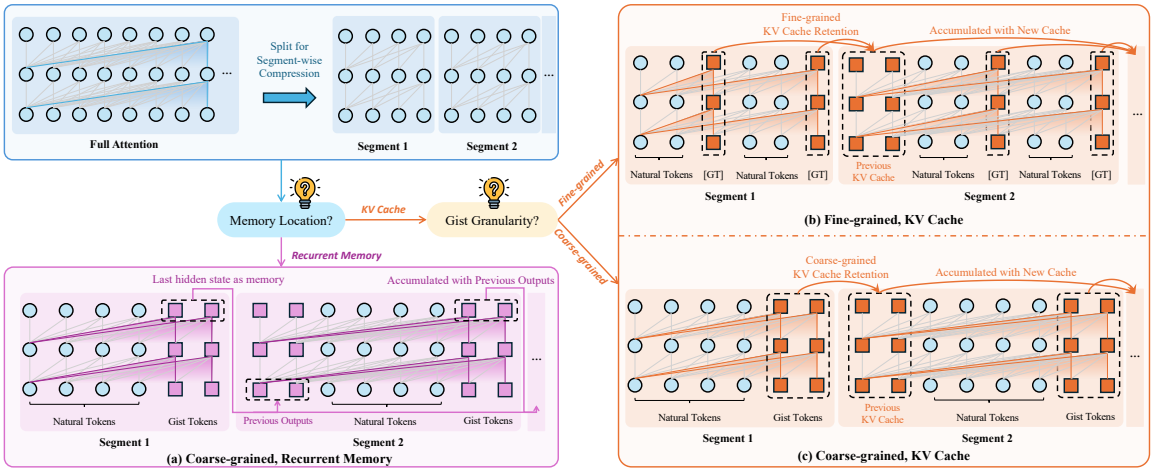

Figure 1: Overview of gist token-based context compression architectures. The purple and orange arrows are used to highlight the maximum range of tokens each gist token can attend to in each segment.

tasks like reranking and synthetic recall, suggesting that while promising, it is prone to severe compression failures in certain scenarios. Regarding *Q2*, we conduct a probing experiment focused on context reconstruction and discover that the compression bottlenecks occur in the gist representations. We further identify three failure patterns resulting from this bottleneck: 1) *lost by the boundary*, where generation degrades near the start of a segment; 2) *lost if surprise*, where unexpected details tend to be ignored if budgets are limited; and 3) *lost along the way*, where compressed models make errors midway for tasks requiring precise recall.

Building on the above findings, we further propose two strategies to enhance the Fine KV architecture for more effective context compression. The first, *fine-grained autoencoding*, adds a weak decoder with an autoencoding loss to reconstruct original token information from gist tokens, ensuring efficient and accurate compression. The second, *segment-wise token importance estimation*, adjusts loss weights based on a token's dependency on the compressed context, dynamically optimizing tokens that require more contextual understanding. Experiments show that both strategies significantly improve model performance, with joint optimization achieving the best results.

The contributions of this work are as follows:

- We propose a unified framework for categorizing existing gist-based model architectures and conduct comprehensive experiments to evaluate their effectiveness. (§2)
- We show that gist-based models achieve effective compression on many tasks but still face challenges in particular scenarios. (§3)
- We identify three critical failure patterns arising from compression bottlenecks, offering valuable insights into the limitations of current gist-based compression methods. (§4)
- We propose two strategies: fine-grained autoencoding and segment-wise token importance estimation, which effectively mitigate these bottlenecks and enhance model performance. (§5)

2 Preliminaries

Gist token-based context compression reduces KV cache by using some special tokens, which are referred to as gists, to represent the full context. The number of special tokens is much fewer than that of the full context, leading to lower memory usage. While many previous work studies compressing the full prompt at once (Mu et al., 2023; Ge et al., 2024b), we focus on a generalized scenario that dynamically compresses and generates context on the fly, as such a setting holds promise for broader general-purpose tasks. To this end, we provide a unified perspective to analyze and understand existing architectures.

Figure 1 illustrates an overview of gist-based context compression methods.[2] We take a segment-wise approach that splits the input sequence into segments and iteratively applies compression for each segment. Assuming an input sequence $X = [x_1, \ldots, x_n]$, it is divided into segments of fixed length L, where the i-th segment is represented as $S_i = [x_{(i-1)\cdot L+1}, \ldots, x_{(i-1)\cdot L+L}]$. When processing the i-th segment, the model accumulates all previously compressed information and generates

[2]For clarity and ease of understanding, we further provide detailed examples for each architecture in Appendix A.

new compressed representations as the memory for later processing:

$$\hat{G}_{<(i+1)} \leftarrow \text{LLM}([\hat{G}_{<i}, \text{Insert}(S_i, G_i)]) \quad (1)$$

Here, $G_i = [g_1, \ldots, g_t]$ are new gist tokens inserted into the i-th segment, and $\hat{G}_i$ are compressed context representations preceding this segment. The function Insert$(\cdot)$ denotes the insertion of gist tokens into the input sequence. This procedure effectively compresses the information of L tokens into t tokens, achieving a compression ratio of L/t. For example, with a compression ratio of 4, every four raw tokens can be replaced by one gist token on average, thereby reaching a 75% reduction in memory usage. Following this equation, existing architectures can be categorized along two dimensions: "memory location" and "gist granularity".

Memory Location After the forward pass of each segment, we can choose to store either the last hidden states of the gist tokens or their KV cache as memory. Opting for the last hidden states is commonly referred to as "recurrent memory", which serves as input embeddings to deliver compressed context to subsequent segments. Note that this design can be viewed as a segment-wise RNN, and typical representatives include RMT (Bulatov et al., 2022) and AutoCompressors (Chevalier et al., 2023). Alternatively, the KV cache of the gist tokens can be directly reused as the memory to avoid extra computations, and this shares the same design as in sparse attention. Typical representatives of the KV approach include Gist (Mu et al., 2023), Landmark (Mohtashami and Jaggi, 2023), and Activation Beacon (Zhang et al., 2024a).

Gist Granularity The Insert$(\cdot)$ function in the equation can be implemented in two ways: (1) Coarse-grained: Gist tokens are appended after all raw tokens, allowing each gist token to attend to the entire segment and all preceding contexts, which is the scheme adopted in most previous works; (2) Fine-grained: Gist tokens are evenly inserted among the raw tokens, enabling each gist token to focus on a specific context, which is investigated in Activation Beacon (Zhang et al., 2024a). Besides, this design can also enhance language modeling through an implicit chain-of-thought mechanism.

Notably, the combination of recurrent memory and fine-grained gist tokens is practically infeasible, since it requires too many non-parallelizable forward passes within a segment. Therefore, we mainly explore the remaining three combinations in this work, as illustrated in Figure 1.

	Coarse-grained	Fine-grained
Recurrent Memory	AutoCompressors (Chevalier et al., 2023)	-
KV Cache	Gist (Mu et al., 2023)	Beacon (Zhang et al., 2024a)

Table 1: Representative works of three combinations.

Categorization We organize the two dimensions into Table 1 and list representative works. These methods commonly exhibit the following two significant characteristics: (1) They introduce new special tokens for compression, requiring continued training to adapt to new patterns. (2) Any compressed original token will be attended to by at least one gist token, thus ensuring that the model can at least in terms of expressive capability access all raw tokens. These effectively distinguish them from token-eviction methods. We will elaborate further on this distinction in the related work (§6).

3 Can Gist Tokens Replace Full Attention in an Efficient and Effective Way?

3.1 Experimental Setup

Training Recipes In our main experiments, we perform continued training on the base models using a general-purpose corpus to analyze their intrinsic context compression capabilities. To avoid potential confounding effects from techniques like supervised fine-tuning, we focus exclusively on the base models rather than the SFT ones.[3] Specifically, we use Llama3.1-8B (Meta-Llama, 2024) as our base model in the main text, given its widespread recognition and adoption in the community.[4] We use the SlimPajama dataset and follow the processing procedure of Fu et al. (2024), by upsampling long sequences and ultimately obtaining 3B tokens for training. Further training details are provided in Appendix B.

Evaluation Tasks We perform extensive experiments, covering a wide range of tasks: (1) *Language modeling*, for which we evaluate perplexity on PG19 (Rae et al., 2020), Proof-Pile (Zhangir Azerbayev), and CodeParrot (CodeParrot); (2) *Weak Context-dependent Tasks*[5], for

[3] Extra analysis of SFT is shown in Appendix D.4.

[4] We also include the results of Qwen2-7B (Qwen-Team, 2024) and Llama3.2-3B in the appendix (D.2 & D.3).

[5] These tasks do not inherently require long contexts. We increase their context length by adding weak-dependent examples to achieve at least one compression.

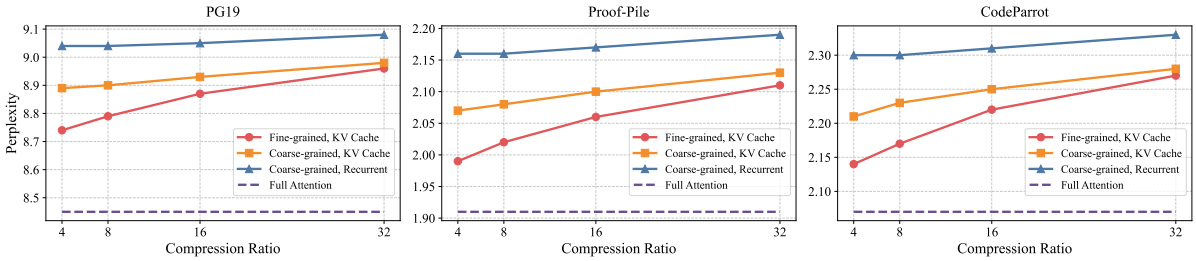

Figure 2: Comparisons of different compression methods on perplexity evaluation for language modeling.

which we evaluate four tasks with MMLU-Pro (Wang et al., 2024), GSM8K (Cobbe et al., 2021), HellaSwag (Zellers et al., 2019), and BBH (Suzgun et al., 2023), to evaluate the model's abilities in knowledge, mathematics, common sense, and comprehensive reasoning, respectively; (3) *Long Context Tasks*, which thoroughly assess the model's handling of long texts and we select HELMET (Yen et al., 2024) as our main benchmark. It includes portions from popular long-text benchmarks such as RULER (Hsieh et al., 2024) and ∞Bench (Zhang et al., 2024b). We also adopt 2-shot demonstrations to ensure a robust evaluation of long-context performance. Further details on the datasets and metrics are provided in Appendix C.

Our study concentrates on the analysis of gist token-based methods. To maintain research focus, we provide extra quantitative comparison with popular token-eviction methods in Appendix E.

3.2 Overall Performance Comparisons

Language Modeling As shown in Figure 2, the differences between the architectures are clear and consistent across all datasets. Full attention outperforms all methods that compress contexts. Among the compression-enhanced architectures, fine-grained compression delivers better performance than coarse-grained, and KV cache performs better than recurrent memory. Note that the absolute differences in perplexity are small; for example, with a compression ratio of 4, the gap between the fine-grained KV cache and the full attention on Proof-Pile is only 0.1.

Weak Context-dependent Tasks As shown in Table 2,[6] among four datasets, full attention shows a clear advantage only on the BBH dataset, which involves some complex reasoning tasks. In the BBH dataset, reasoning paths can usually extend over several hundred tokens. Long-form reasoning within compressed contexts frequently encounters challenges, such as generating content that spans multiple segments, which results in the accumulation of substantial inaccuracies during the process. This severely impacts the final output. However, in the other three datasets, despite the diversity of task types, the reasoning paths are typically only dozens of tokens long, which explains why compression models maintain most of the performance.

Ratio	Type	MMLU-Pro	BBH	GSM8K	HellaSwag
-	Full Attention	34.1	64.8	51.2	82.8
4	Coarse-Rec	34.1	53.8	50.3	81.9
	Coarse-KV	35.3	58.1	48.7	82.3
	Fine-KV	33.9	59.2	52.2	82.5
8	Coarse-Rec	34.1	54.6	51.9	82.0
	Coarse-KV	35.6	56.1	49.0	82.2
	Fine-KV	34.6	56.8	51.9	82.5
16	Coarse-Rec	34.1	53.2	50.0	81.9
	Coarse-KV	35.6	55.7	50.1	82.2
	Fine-KV	34.3	56.0	51.7	82.2

Table 2: Performance on weak context-dependent tasks.

Long Context Tasks Table 3 presents the results, where we have the following findings: **(1) Higher compression ratio leads to lower performance.** While Fine-KV can achieve comparable performance to full attention in some tasks at lower compression ratios (e.g., 4), it struggles to maintain this level of performance at higher ratios. **(2) The extent of performance degradation in compressed models varies significantly across different types of tasks.** For tasks where the required information is somewhat fuzzy (e.g., Summarization), or where the query is closely related to the general topics of the context (e.g., RAG and LongQA), compression does not noticeably affect the performance. For many-shot ICL, which requires almost the full context, the fine-grained KV cache can maintain performance comparable to full attention even at low compression rates. However, in tasks that demand precise rephrasing or involve highly complex multi-hop reasoning, such as Rerank,[7] none of the compressed models perform on par with full at-

[6] Contexts are compressed at least once here. Additional results in the short-context setting can be found in Appendix D.1

[7] This task needs $O(n)$ to evaluate each document, and then sort these documents with $O(n \log n)$ on average.

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	61.8	39.9	41.6	62.3	93.9	23.8	66.1	55.6
	Full Attention, Finetune	61.7	38.5	42.3	60.0	91.0	24.1	65.7	54.7
4	Coarse-grained, Recurrent	49.9	2.1	35.2	29.4	11.2	18.2	59.3	29.3
	Coarse-grained, KV Cache	51.7	5.2	33.9	36.0	14.2	17.6	57.8	30.9
	Fine-grained, KV Cache	**60.6**	**23.4**	**40.3**	**70.6**	**40.6**	**21.0**	**63.0**	**46.2**
8	Coarse-grained, Recurrent	49.8	1.3	36.0	25.9	11.2	**17.7**	58.6	28.6
	Coarse-grained, KV Cache	50.8	3.8	36.5	33.6	13.5	16.1	57.2	30.2
	Fine-grained, KV Cache	**57.6**	**14.5**	**40.2**	**68.1**	**26.9**	16.7	**60.7**	**40.7**
16	Coarse-grained, Recurrent	49.9	1.4	34.9	20.8	11.2	**17.8**	57.5	27.6
	Coarse-grained, KV Cache	50.2	4.4	34.2	29.1	13.1	16.7	58.1	29.4
	Fine-grained, KV Cache	**55.4**	**10.0**	**40.4**	**49.3**	**13.8**	16.3	**59.2**	**34.9**

Table 3: Performance comparison on long context tasks. **Bold** indicates best results along the same ratio.

tention. **(3) Coarse-grained methods appear to struggle in fully utilizing the available memory budget.** Despite having the same memory budget, the Fine-KV's performance decreases systematically as the compression rate increases, whereas coarse-grained methods show consistently poor performance across different ratios. The trends observed in perplexity evaluation support this finding, suggesting that coarse-grained gist placement is less effective at learning how to optimize the memory budget for compression.

4 Understanding Why and How Compression Fails

Previous results show that gist token-based context compression exhibits a discernible performance gap compared to full attention, particularly in tasks like synthetic recall that require exact rehearsal. This suggests the presence of a "compression bottleneck" that prevents the language model from treating gist tokens as equivalent to uncompressed context. In this section, we conduct a probing experiment to test whether the original context can be faithfully recovered from gist tokens and examine three critical failure modes arising from it.

4.1 Compression Bottleneck Probing

Experimental Setting We adopt the concept of autoencoder to investigate the quality of compressed representations in gist tokens. For this experiment, we use the Fine-KV architecture, which is the most effective compression architecture according to previous results. We evaluate whether each gist token completely stores the contextual information of its corresponding snippet by training a probing decoder to recover the corresponding token sequence. We examine two decoders: an 8B model that inherits the full pre-trained parameters and a model with only one transformer layer. This allows us to explore the compression quality from the perspective of decoder capacities.

Decoder Type	Train Loss	Reconstruction Accuracy 4	8	16	32
Weak	2.64	53.9%	19.2%	9.6%	5.1%
Strong	2.01	77.3%	39.9%	19.3%	10.0%

Table 4: Performance of reconstruction.

Results In Table 4, we report the training loss after 2K training steps for two models, along with their token-level reconstruction accuracy on the PG19 dataset. Although the full model demonstrates superior performance, it still exhibits significant shortcomings in decoding the information within gist tokens. Under high compression ratios, the model's accuracy even falls below 20%, indicating that it can only retain fuzzy content rather than remember the precise details from the original context. Ideally, copying a small set of recent tokens should be an easy task, yet probing experiments reveal poor performance. This suggests that the representations of current gist token memory impose a severe compression bottleneck, limiting the model's capacity to extract and utilize contextual information effectively.

4.2 Failure Pattern Observations

The compression bottleneck may evolve into specific failure patterns. We highlight three representative and interesting patterns:

Lost by the boundary This discovery stems from an analysis of token-level perplexity distribution. As illustrated in Figure 3, we compute the average perplexity of the tokens at each position within individual segments, excluding the first segment since it lacks gist tokens as contextual input. The results reveal that, while token perplexity in the full attention model remains relatively uniform across positions, the compressed model exhibits a

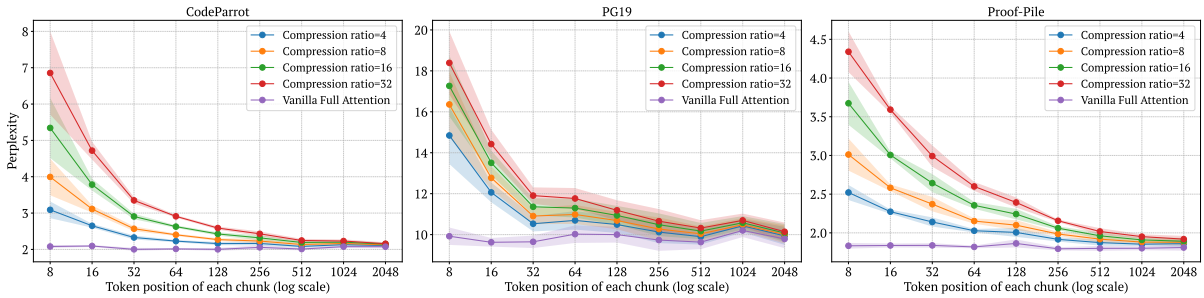

Figure 3: Average Perplexity of tokens in different positions among segments.

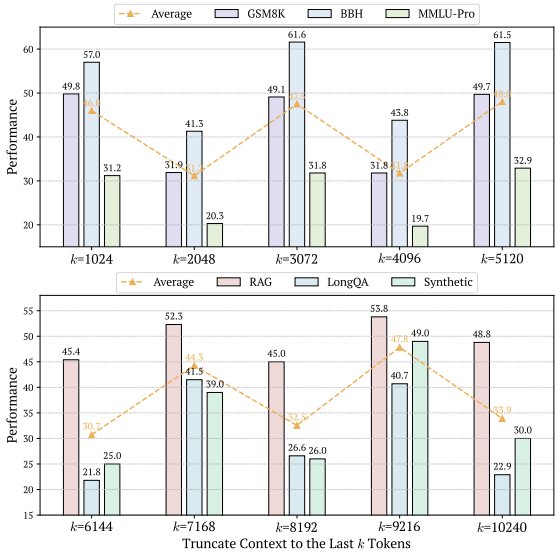

Figure 4: Performance on different tasks while truncating context to the last k tokens. When k is a multiple of 2048, the model will generate near the boundary.

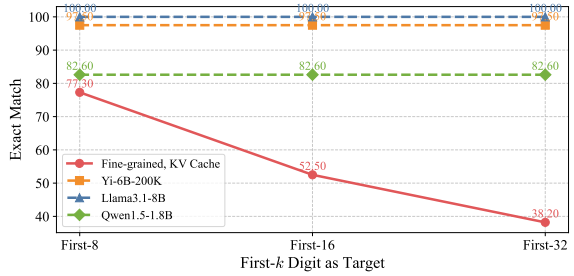

Figure 5: Performance on the 32-digit uuid recall task. We report the exact match rates of various first-k digits.

clear pattern of higher perplexity at the start of the segment and lower perplexity toward the end.

Furthermore, we evaluated the impact on generation tasks by truncating the context to a specific length. As shown in Figure 4, with a segment length set to 2K, the performance when generation starts at the beginning of a segment is substantially worse compared to the case when generation starts from the middle of a segment. This indicates that the segment boundary effects influence not only the accuracy of reading specific information but also the model's overall language modeling capability.

Needle Type	Rel.	Compression Ratio 4	8	16	32
Word	✓	89.8(+0.0)	50.7(+0.0)	26.0(+0.0)	19.6(+0.0)
	✗	89.6(-0.2)	35.8(-14.9)	18.0(-8.0)	16.8(-2.8)
Number	✓	84.5(+0.0)	69.2(+0.0)	26.3(+0.0)	17.2(+0.0)
	✗	84.4(-0.1)	59.0(-10.2)	20.9(-5.7)	16.6(-0.6)

Table 5: Performance on synthetic recall task (PopQA).

Lost if surprise We find that under constrained memory budgets, the model tends to prioritize retaining detailed information that closely aligns with the overarching theme of the context. To validate this, we construct a synthetic dataset[8] with different configurations based on the PopQA dataset from the RAG task, as it provides explicit question subjects, and most documents are typically related to the same subject. We randomly insert a "needle" between sentences in the gold document, formatted as: "{subj}'s special {needle_type} is {needle_content}". Here, {subj} can either be the original subject or "Mr. Tree", while {needle_type} can be either "food" or an 8-digit number. When {subj} is the original subject, we consider the needle to be relevant to the theme of most of the context; otherwise, it is surprising and unrelated. All needles are transformed into compressed gist tokens during the model's decoding stage. As shown in Table 5, our experimental results reveal significant performance differences in both needle types when altering only the subject of a single sentence. This indicates that the successful retrieval of compressed information is associated with its relevance to the context. An "unexpected" information is more likely to be lost during compression.

Lost along the way We notice that compression-enhanced architectures struggle to recover exact rehearsal effectively. When dealing with a relatively long "needle", the compression process can scatter critical information across multiple gist tokens. Consequently, even if the model identifies

[8]We provide an example for clarity in Table 21

the beginning of the target information, it risks losing track during subsequent steps of generation. To validate this observation, we conducted a recall experiment using 32-digit UUIDs, comparing the performance of full attention models against compressed models, and analyzed their accuracy across prefixes of varying lengths. As illustrated in Figure 5, the replication accuracy of full attention models remains stable regardless of prefix length, suggesting that once the starting point is identified, copying the rest of the content is straightforward. In contrast, compressed models show a significant drop in accuracy, decreasing to less than half of the original as the prefix extends from the first four digits to all 32 digits. This finding highlights the reduced copying reliability associated with compressed representations.

5 Mitigating Compression Flaws

5.1 Methodology

Building on these findings, we have identified critical shortcomings in the current architecture's context compression. In this section, we propose two effective learning strategies to address them.

Fine-grained Autoencoding (AE) The probing experiments in Section 4 indicate that the compressed representations of current gist tokens struggle to reconstruct the original content. To address this issue, we introduce an additional autoencoding loss during training to explicitly encourage the retention of the original contextual information. Different from ICAE (Ge et al., 2024b), we require each gist token to be responsible for a specific snippet. Following the mainstream conclusion in autoencoding research that weak decoders help learn better representations (Lu et al., 2021), we adopt a single-layer transformer as the decoder. For each gist token g_i^{kv}, the objective is to reconstruct the original token sequence between the current and previous gist tokens. The input for this task is:

$$[g_i^{kv}, [\text{ae}]_r, x_1, \ldots, x_r],$$

where $[\text{ae}]_r$ is a special token to prompt model to reconstruct r tokens (i.e., x_1 to x_r). The auxiliary loss of autoencoding is similarly defined in an autoregressive way[9]:

$$\mathcal{L}_{\text{ae}} = \frac{1}{N}\frac{1}{r}\sum_{i=1}^{N}\sum_{j=1}^{r}\log P_\theta(x_j|g_i^{kv}, [\text{ae}]_r, x_{<j}).$$

[9]This autoencoding loss is added to the original language modeling loss with a weight α during training.

Segment-wise Token Importance Estimation (TIE) Another approach to promote compression is to adjust the loss weights of different tokens, since each token depends on the context in different degrees. We hypothesize that the importance of a token is determined by the modeling difficulty it presents during segment-wise compression. The more a token relies on the compressed gist context for prediction, the more effort should be dedicated to learning it. Inspired by LongPPL (Fang et al., 2024), we estimate the reliance of each token (x_i) on the gist context and allocate a tailored learning weight w_i accordingly:

$$\text{Diff}(x_i) = \min(\log\frac{P_\theta(x_i|x_{<i}^{\text{seg}})}{P_\theta(x_i|x_{<i}^{\text{full}})}, \gamma),$$

$$w_i = \frac{e^{\text{Diff}(x_i)}}{\sum_{j=1}^{N} e^{\text{Diff}(x_j)}}.$$

Here, P_θ denotes the original language model, $x_{<i}^{\text{seg}}$ denotes the preceding tokens only in the current segment, and $x_{<i}^{\text{full}}$ denotes the full context, including tokens in previous segments. This reliance is quantified by analyzing the difference in modeling probabilities when the token attends to the full context versus the local segment alone.

5.2 Experiments

We evaluate the training efficiency detailed in Appendix F, which shows that our strategies are effectively scalable to continued pre-training.

Boundary Effect Test Previous results show that gist-based models demonstrate strong performance on weak context-dependent tasks but are severely constrained by the "lost by the boundary" phenomenon. We test two improved methods under the same experimental conditions in Section 4, with the results presented in Table 7. Both methods significantly enhance performance in boundary regions, particularly on the BBH dataset, which involves tasks requiring long-form reasoning. This improvement may be attributed to their ability to reduce the accumulation of errors during the generation process. While these methods do not completely eliminate the boundary effect, they offer promising strategies for mitigating its impact.

Long Context Tasks Table 6 highlights that both methods consistently enhance the model's performance on long-context tasks, particularly under low compression ratios. Key observations include: (1) For tasks where the performance gap between

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	61.8	39.9	41.6	62.3	93.9	23.8	66.1	55.6
4	Fine-grained, KV Cache	60.6 (+0.0)	23.4 (+0.0)	40.3 (+0.0)	70.6 (+0.0)	40.6 (+0.0)	21.0 (+0.0)	62.0 (+0.0)	46.1 (+0.0)
	+ Fine-grained AE	60.9 (+0.3)	27.4 (+4.0)	40.8 (+0.5)	72.0 (+1.4)	62.0 (+21.4)	22.3 (+1.3)	62.9 (+0.9)	49.8 (+3.7)
	+ Segment-wise TIE	60.4 (-0.2)	27.0 (+3.6)	41.2 (+0.9)	72.7 (+2.1)	54.3 (+13.7)	20.2 (-0.8)	62.1 (+0.1)	48.3 (+2.2)
	+ Both Strategies	61.1 (+0.5)	27.4 (+4.0)	40.3 (+0.0)	75.0 (+4.4)	62.1 (+21.5)	22.2 (+1.2)	62.9 (+0.9)	**50.1** (+4.0)
8	Fine-grained, KV Cache	57.6 (+0.0)	14.5 (+0.0)	40.2 (+0.0)	68.1 (+0.0)	26.9 (+0.0)	16.7 (+0.0)	60.7 (+0.0)	40.7 (+0.0)
	+ Fine-grained AE	58.3 (+0.7)	15.6 (+0.9)	39.8 (-0.4)	68.7 (+0.6)	34.8 (+7.9)	18.5 (+1.8)	61.3 (+0.6)	42.4 (+1.7)
	+ Segment-wise TIE	58.1 (+0.4)	17.6 (+3.1)	40.0 (-0.2)	70.0 (+1.9)	30.2 (+3.3)	17.7 (+1.0)	60.7 (+0.0)	42.0 (+1.3)
	+ Both Strategies	58.3 (+0.7)	19.7 (+5.2)	40.4 (+0.0)	70.7 (+2.6)	35.2 (+8.9)	19.5 (+2.8)	61.4 (+0.7)	**43.6** (+2.9)
16	Fine-grained, KV Cache	55.4 (+0.0)	10.0 (+0.0)	40.4 (+0.0)	49.3 (+0.0)	13.8 (+0.0)	16.3 (+0.0)	59.2 (+0.0)	34.9 (+0.0)
	+ Fine-grained AE	55.6 (+0.2)	11.3 (+1.3)	40.4 (+0.0)	47.1 (+0.3)	14.7 (+0.9)	16.2 (-0.1)	59.6 (+0.4)	35.0 (+0.1)
	+ Segment-wise TIE	55.6 (+0.2)	10.4 (+0.4)	40.7 (+0.3)	55.5 (+8.4)	14.8 (+1.0)	15.3 (-1.0)	58.1 (-1.1)	35.7 (+0.8)
	+ Both Strategies	56.3 (+0.9)	12.7 (+2.7)	41.7 (+1.3)	56.3 (+7.0)	14.9 (+1.1)	15.7 (-0.6)	59.6 (+0.4)	**36.7** (+1.8)

Table 6: Performance comparisons using our methods, with the best "average" results bolded for clarity.

k	Model	MMLU-Pro	BBH	GSM8K
2048	Fine-grained KV	20.3 (+0.0)	41.3 (+0.0)	31.9 (+0.0)
	+ Fine-grained AE	23.4 (+3.1)	47.8 (+6.5)	34.3 (+2.4)
	+ Segment-wise TIE	22.9 (+2.6)	46.3 (+5.0)	32.3 (+2.0)
4096	Fine-grained KV	19.7 (+0.0)	43.8 (+0.0)	31.8 (+0.0)
	+ Fine-grained AE	22.5 (+2.8)	51.0 (+7.2)	35.1 (+3.3)
	+ Segment-wise TIE	22.9 (+3.2)	50.8 (+7.0)	34.7 (+2.9)

Table 7: Improvements of our mitigating methods on the "lost by the boundary" problem.

the compression-enhanced model and full attention is relatively small (e.g., RAG and LongQA), both methods maintain excellent performance without negative impacts. For the many-shot ICL task, they even demonstrate continuous improvements. (2) For tasks where the original architectures struggle, such as rerank and synthetic recall, both methods deliver remarkable performance gains. For instance, under a compression ratio of 4, the improvements on the synthetic recall task reach as high as 52.7% and 33.7%, respectively. These indicate that our methods can effectively enhance the model to read context information from gist tokens.

6 Related Work

KV Cache Compression Recent work has explored KV cache optimization at the *layer*, *head*, *token*, and *tensor* levels. *Layer*-level methods merge caches across layers using inter-layer similarities (Brandon et al., 2024; Sun et al., 2024; Wu and Tu, 2024; Liu et al., 2024a). *Head*-level techniques allow multiple query heads to share key-value pairs (Ainslie et al., 2023; Shazeer, 2019). *Tensor*-level approaches, such as low-rank approximations, compress caches into compact representations (DeepSeek-AI, 2024), while quantization reduces precision for memory savings (Liu et al., 2024c). *Token*-level methods preserve only critical tokens, including learnable tokens (Mu et al., 2023; Ge et al., 2024b; Qin and Durme, 2023; Mohtashami and Jaggi, 2023; Chevalier et al., 2023; Zhang et al., 2024a), token eviction (Xiao et al., 2024c; Liu et al., 2023; Ge et al., 2024a), external memory (Xiao et al., 2024a), and hard selection (Li et al., 2023; Qin et al., 2024; Jiang et al., 2024b). In this work, we focus on the direction that introduces a few learnable special tokens to replace the previous full context.

Sparse Attention Researchers have been exploring efficient alternatives of full attention (Beltagy et al., 2020; Zaheer et al., 2020; Kitaev et al., 2020; Zhou et al., 2022; Tay et al., 2020). Recently, it has been widely observed that LLMs naturally exhibit significant sparse attention patterns, especially in long-form texts (Jiang et al., 2024a). To leverage such characteristics, researchers have developed heuristic or learnable sparsification strategies that achieve significant speedup while maintaining reliable performance (Jiang et al., 2024a; Xiao et al., 2024b). The gist token-based context compression approach can be regarded as a special case of sparse attention with a segment-wise approach (Chevalier et al., 2023; Zhang et al., 2024a): where full attention is employed within each segment.

7 Conclusion

Our comprehensive evaluation shows that while gist-based context compression shows promise as an alternative to full attention in many tasks, it still falls short in specific scenarios. Through carefully designed probing experiments, we identify critical compression bottlenecks and typical failure modes. Furthermore, we propose two effective strategies that significantly enhance compression performance. These findings offer new insights and directions for advancing context compression techniques in the future.

Limitations

Constrained by our available computational resources, we are able to train long-text large language models with sizes up to 7/8B parameters in a 16K context window. Larger models (e.g., Llama3.1-70B) typically have more layers, which enables them to offer greater memory capacity and stronger reading capabilities under the same compression ratio when using gist token-based compression. Thus, such larger models may offer advantages in reducing performance degradation, but this still needs to be verified in future studies.

Ethical Discussion

This study focuses on the performance of gist token-based context compression techniques, without introducing explicitly designed features that could directly influence the cognition of language models. We select widely recognized and validated public training datasets. This can minimize the risk of injecting new biases or toxic data. These datasets are typically subjected to rigorous review and curation, ensuring balanced and stable data distributions. As a result, they help mitigate the impact of harmful information on the model's learning process and prevent significant distortions in its cognitive and decision-making patterns.

Acknowledgement

This work was supported by Beijing Municipal Science and Technology Project No. Z231100010323009, National Science and Technology Major Project No. 2022ZD0120103, National Natural Science Foundation of China No. 62272467, Beijing Natural Science Foundation No. L233008, and the fund for building world-class universities (disciplines) of Renmin University of China. The work was partially done at the Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137. Association for Computational Linguistics.

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

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. 2024. Reducing transformer key-value cache size with cross-layer attention. *CoRR*, abs/2405.12981.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. 2022. Recurrent memory transformer. *CoRR*, abs/2207.06881.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *CoRR*, abs/2306.15595.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2024. Longlora: Efficient fine-tuning of long-context large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

CodeParrot. https://huggingface.co/codeparrot/codeparrot.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *CoRR*, abs/2405.04434.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l_2 norm-based strategy for KV cache compression. *CoRR*, abs/2406.11430.

Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding,

and Yisen Wang. 2024. What is wrong with perplexity for long-context language modeling? *CoRR*, abs/2410.23771.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2024. Ada-kv: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *CoRR*, abs/2407.11550.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128k context. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2024. How to train long-context language models (effectively). *CoRR*, abs/2410.02660.

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

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024a. Model tells you what to discard: Adaptive KV cache compression for llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024b. 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.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: what's the real context size of your long-context language models? *CoRR*, abs/2404.06654.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024a. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *CoRR*, abs/2407.02490.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024b. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 1658–1677. Association for Computational Linguistics.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wojciech Kryscinski, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. 2022. BOOKSUM: A collection of datasets for long-form narrative summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 6536–6558. Association for Computational Linguistics.

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.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024a. Minicache: KV cache compression in depth dimension for large language models. *CoRR*, abs/2405.14366.

Tianyang Liu, Canwen Xu, and Julian J. McAuley. 2024b. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Shuqi Lu, Di He, Chenyan Xiong, Guolin Ke, Waleed Malik, Zhicheng Dou, Paul Bennett, Tie-Yan Liu, and Arnold Overwijk. 2021. Less is more: Pretrain a strong siamese encoder for dense text retrieval using a weak decoder. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 2780–2791. Association for Computational Linguistics.

Meta-Llama. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Amirkeivan Mohtashami and Martin Jaggi. 2023. Random-access infinite context length for transformers. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jesse Mu, Xiang Li, and Noah D. Goodman. 2023. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. Yarn: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. Memorag: Moving towards next-gen RAG via memory-inspired knowledge discovery. *CoRR*, abs/2409.05591.

Guanghui Qin and Benjamin Van Durme. 2023. Nugget: Neural agglomerative embeddings of text. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 28337–28350. PMLR.

Guanghui Qin, Corby Rosset, Ethan C. Chau, Nikhil Rao, and Benjamin Van Durme. 2024. Dodo: Dynamic contextual compression for decoder-only lms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9961–9975. Association for Computational Linguistics.

Qwen-Team. 2024. Qwen2 technical report. *CoRR*, abs/2407.10671.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *CoRR*, abs/2405.05254.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *CoRR*, abs/2009.06732.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *CoRR*, abs/2406.01574.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Haoyi Wu and Kewei Tu. 2024. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 11175–11188. Association for Computational Linguistics.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, Song Han, and Maosong Sun. 2024a. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. *CoRR*, abs/2402.04617.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024b. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. *CoRR*, abs/2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024c. Efficient streaming

language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. 2024. HELMET: how to evaluate long-context language models effectively and thoroughly. *CoRR*, abs/2410.02694.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

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*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024b. ∞bench: Extending long context evaluation beyond 100k tokens. *CoRR*, abs/2402.13718.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024c. A survey on the memory mechanism of large language model based agents. *CoRR*, abs/2404.13501.

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

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Yujia Zhou, Zhicheng Dou, Huaying Yuan, and Zhengyi Ma. 2022. Socialformer: Social network inspired long document modeling for document ranking. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 339–347. ACM.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *CoRR*, abs/2308.07107.

A Gist Token-based Compression Examples

To facilitate understanding for readers unfamiliar with this type of method, we provide several toy examples here. The following examples are all described based on Equation 1. Specifically, we consider a token sequence $X = [x_1, ..., x_{12}]$ with a total length of 12, and set the segment length to 4. All architectures uniformly adopt a 2:1 compression ratio to achieve a 50% reduction in KV cache. In practice, all gist tokens share the same embedding to accelerate model convergence.

In the following, we will detail the processing flow of three architectures and provide corresponding examples. To avoid repetitive descriptions, we will describe the "*Coarse-grained, Recurrent Memory*" architecture in more detail, while the descriptions of the other two architectures will be appropriately streamlined.

A.1 Coarse-grained, Recurrent Memory

First, the input token sequence is divided into three segments, and each segment will be compressed by gist tokens of half the size (i.e., 2 tokens):

$$\begin{aligned}[S_1, S_2, S_3] &= [[x_1, ..., x_4], ..., [x_9, ..., x_{12}]], \\ [G_1, G_2, G_3] &= [[g_1, g_2], [g_3, g_4], [g_5, g_6]]\end{aligned} \tag{2}$$

Subsequently, using the Insert$(\cdot)$ function, we insert all the gist tokens $G_1 = [g_1, g_2]$ into the first segment in a coarse-grained manner, specifically at the end of the segment:

$$\text{Insert}(S_1, G_1) = [x_1, ..., x_4, g_1, g_2]. \tag{3}$$

After feeding this sequence into the LLM for a forward pass, we obtain the hidden states of g_1 and g_2 at the last layer of the model output, denoted as $\hat{G}_1 = [\hat{g}_1, \hat{g}_2]$. These representations are the compressed representation of the current segment.

Next, we discard the original tokens in the first segment and retain only $\hat{G}_1$ to model the contextual representation of the second segment. Its input is $[\hat{g}_1, \hat{g}_2, x_5, ..., x_8, g_3, g_4]$. Similarly, we input this into the LLM and retain only the representations of g_3 and g_4 at the last layer of the model output (i.e., $\hat{G}_2 = [\hat{g}_3, \hat{g}_4]$).

For the third segment, we accumulate the compressed information from all previous segments $\hat{G}_{<3} = [\hat{g}_1, \hat{g}_2, \hat{g}_3, \hat{g}_4]$, and model the third segment with previous compressed gist context in the same manner.

During the training stage, we retain the original token outputs in each segment so that we can train by calculating the language modeling loss, just like a standard full attention model. In the inference phase, we employ a segmented progressive compression of context to achieve low peak memory usage.

A.2 Coarse-grained, KV Cache

As depicted in the Insert$(\cdot)$ function of Equation 3, gist tokens are still fully inserted at the end of the first segment. Subsequently, this sequence is fed into the LLM for forward computation. It is noteworthy that, differing from "Recurrent Memory" architecture, we do not merely utilize the hidden states of the last layer as input. Instead, we preserve the KV cache of all layers corresponding to g_1 and g_2, forming $\hat{G}_1 = [\hat{g}_1, \hat{g}_2]$. This implies that semantic information is compressed into the KV cache of each layer of gist tokens, offering a greater memory capacity compared to using only a single hidden state.

When processing the second segment, the input is $[\hat{g}_1, \hat{g}_2, x_5, ..., x_8, g_3, g_4]$. Here, $\hat{g}_1$ and $\hat{g}_2$ serve as the prefix KV cache, while the other elements are normal token inputs.

In dealing with the third segment, the KV cache $\hat{G}_{<3}$ corresponding to the gist tokens of the preceding segments is also accumulated to serve as the prefix KV cache context, which is then used to model the tokens of the current segment.

A.3 Fine-grained, KV Cache

Compared to the coarse-grained approach, the key difference in the fine-grained architecture is the placement of gist tokens. The coarse-grained method inserts all gist tokens at the end of a segment, intending to ensure each gist token can capture the complete information of that segment. In contrast, the fine-grained method inserts gist tokens evenly across different positions within the segment. This utilizes the autoregressive property of causal LLMs to precisely define the information scope compressed by each gist token. Within the fine-grained architecture, the Insert$(\cdot)$ of Equation 3 is updated to:

$$\text{Insert}(S_1, G_1) = [x_1, x_2, g_1, x_3, x_4, g_2]. \tag{4}$$

For modeling subsequent segments, we similarly retain only the KV cache corresponding to prior gist tokens as the compressed context.

B Training Details

We train all models using 3B tokens from the upsampled SlimPajama dataset, with document boundaries marked by the eos token. Each model was augmented with 4 sink tokens (Xiao et al., 2024c) to enhance modeling stability. To support dynamic compression ratio assignment, the compression ratio for each data instance is randomly sampled from {4, 8, 16, 32}. The context length of the training data is set to 16K, with a fixed segment length of 2K. The batch size is set to 2M tokens. The learning rate is set to 1e-5, using a cosine lr scheduler that reduces the learning rate to 50% of its highest value in the end. The autoencoding weight α is set to 0.1. Additionally, the first 1% of training steps are allocated for learning rate warmup.

C Evaluation Details

Perplexity The average perplexity is calculated across all data using a 16K-length context window, with a sliding window stride equal to the length of the context window.

Weak Context-dependent Tasks To ensure that the context for each task is compressed at least once, few-shot examples are used to fill the context. The number of examples used for each task is detailed in Table 8. For all tasks except HellaSwag, which selects answers based on the likelihood of candidate answers, the Chain-of-Thought (CoT) reasoning approach is employed to generate answers.

Dataset	#Few-shot demos	Answer acquisition
MMLU-Pro	12	Chain-of-Thought
BBH	8	Chain-of-Thought
GSM8K	16	Chain-of-Thought
HellaSwag	32	Logits

Table 8: Settings of weak context-dependent tasks.

Long Context Tasks The majority of our task configurations are based on Yen et al. (2024) and Gao et al. (2024), with code tasks leveraging RepoBench (Liu et al., 2024b; Bai et al., 2024). We sample up to 1K samples for each dataset, and contexts are constructed under the configs of a max length of 16K. The datasets' descriptions are presented in Table 9. More configuration details like RAG setting can be found in HELMET's official GitHub repo[10]. We apply greedy decoding to all generation tasks for stability.

Category	Tasks	Metrics
RAG	NQ	SubEM
	TriviaQA	SubEM
	PopQA	SubEM
	HotpotQA	SumEM
Rerank	MS Marco	NDCG@10
Long-doc QA	∞Bench QA	ROUGE Recall
	∞Bench MC	Accuracy
Many-shot ICL	TREC Coarse	Accuracy
	TREC Fine	Accuracy
	NLU	Accuracy
	BANKING77	Accuracy
	CLINIC150	Accuracy
Synthetic recall	JSON KV	SubEM
	RULER MK Needle	SubEM
	RULER MK UUID	SubEM
	RULER MV	SubEM
Summ.	∞Bench Sum	ROUGE-Sum F1
	Multi-LexSum	ROUGE-Sum F1
Code	RepoBench	Edit Distance

Table 9: Details of long context tasks.

Type	MMLU-Pro	BBH	GSM8K	HellaSwag
Full Attention	35.1	59.0	50.9	79.8
Coarse, Rec	34.8	59.2	50.4	79.3
Coarse, KV	35.1	58.5	51.6	79.2
Fine, KV	35.0	59.5	50.1	79.5

Table 10: Performance of short context tasks.

D More Experimental Results

D.1 Results in the Short Context Setting

We report model performance in the short context setting in Table 10, in which 2-shot demos are applied and contexts are not compressed. The results indicate that short-context capabilities are not affected by learning compression.

D.2 Performance of Qwen2-7B

In addition to LLAMA3.1-8B, we also conduct a full set of experiments on another widely acknowledged model, QWEN2-7B. The results are shown in Table 11.

D.3 Performance of Llama-3.2-3B

In addition to the 7-8B models, we also evaluate the LLAMA-3.2-3B model to explore the compression performance of smaller-scale models. Given that LLAMA-3.2-3B shares the same vocabulary as LLAMA-3.1-8B, we adopt an identical training

[10]https://github.com/princeton-nlp/HELMET

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	56.2	26.6	44.5	67.1	81.8	19.0	64.6	51.4
4	Coarse-grained, Recurrent	44.1	0.9	35.6	27.9	12.1	19.3	56.9	28.1
	Coarse-grained, KV Cache	45.4	1.6	36.2	29.8	12.4	17.8	59.4	29.2
	Fine-grained, KV Cache	54.8	10.6	43.8	67.5	15.5	18.2	59.4	**38.9**
8	Coarse-grained, Recurrent	49.8	1.3	36.0	25.9	11.2	17.7	58.6	28.6
	Coarse-grained, KV Cache	44.8	0.5	39.3	28.5	12.3	18.1	59.4	28.9
	Fine-grained, KV Cache	52.0	5.0	44.2	62.7	11.6	17.9	61.7	**36.4**
16	Coarse-grained, Recurrent	49.9	1.4	34.9	20.8	11.2	17.8	57.5	27.6
	Coarse-grained, KV Cache	45.1	0.9	38.6	27.9	12.2	17.8	58.7	28.7
	Fine-grained, KV Cache	49.5	3.1	42.2	44.5	11.7	16.9	59.6	**32.5**
32	Coarse-grained, Recurrent	44.2	2.4	34.1	27.5	11.5	18.5	57.3	27.9
	Coarse-grained, KV Cache	45.0	1.1	37.1	23.6	12.2	17.6	57.9	27.8
	Fine-grained, KV Cache	47.5	1.7	40.6	36.9	12.1	16.8	59.5	**30.8**

Table 11: Long context performance based on QWEN2-7B.

recipe for training this model. To accelerate the experimental process, we exclusively employ a compression ratio of 4. As illustrated in Table 12, the performance patterns of LLAMA-3.2-3B across various tasks align with the conclusions drawn in the main text. However, the performance gap with full-attention models slightly widens on tasks where the 8B model originally excels (e.g., RAG and LongQA).

D.4 Results of Supervised Fine-tuning

Supervised Fine-tuning (SFT) is a critical factor influencing model performance on downstream tasks. Gist token-based context compression models often struggle with certain tasks (e.g., synthetic ones), which may be attributed to the low proportion of long-dependency data in the general-purpose continue-training corpus. To investigate the effect of high-quality SFT data on the model's compression ability, we fine-tune the LLAMA3.1-8B with the Fine-KV architecture. The training data consists of LongAlpaca (Chen et al., 2024), BookSum (Kryscinski et al., 2022), and synthetic data from (Zhang et al., 2024a). We then evaluate its performance on long-context tasks. Table 13 presents the detailed results: the fine-tuned model shows significant gains in the previously weakest task (i.e., synthetic recall), while maintaining its performance on tasks where it already excelled. This suggests that long-range supervised signals effectively enhance the ability of gist tokens to preserve precise information in dense memory. Thus, high-quality SFT data containing long-distance dependencies is not only beneficial but potentially essential for the compression model.

E Quantitative Comparison with Token-eviction Methods

As the primary objective of this research is to delve into gist token-based compression methods, the comparison with token-eviction methods is somewhat beyond the scope of the main study. Therefore, we include this section in the Appendix. This section will quantitatively compare these two types of methods from the dimensions of "Performance" and "Efficiency". All experiments are based on the LLAMA-3.1-8B model. We select "Fine-grained, KV cache" as the representative of gist token-based methods, and implement all token-eviction methods using Huggingface Transformers and widely-used KVPress[11].

Performance Consistent with our experiments in the main text, we use HELMET (Yen et al., 2024) to evaluate all the models. We select popular token-eviction methods such as StreamingLLM (Xiao et al., 2024c), Knorm (Devoto et al., 2024), SnapKV (Li et al., 2024), and AdaKV (Ada-SnapKV) (Feng et al., 2024) for comparison. Following Feng et al. (2024), we apply question-agnostic evaluation for all methods, in which context compression is performed without relying on the question itself. This is more in line with practical application scenarios (e.g., multi-turn conversations) and ensures the fairness of the experiments. Considering that Feng et al. (2024) has pointed out that the performance of token-eviction methods decreases significantly when the KV cache is reduced by 50%, we choose to evaluate at the lowest compression ratio (i.e., compression ratio=4).

As shown in the results of Table 18, the gist

[11] https://github.com/NVIDIA/kvpress

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	57.9	33.1	36.4	72.6	89.1	22.7	64.5	53.7
4	Fine-grained, KV Cache	54.8	18.2	33.5	67.9	48.8	18.0	62.5	43.4

Table 12: Long context performance of models based on LLAMA-3.2-3B. **Bold** indicates the best result along the same compression ratio.

Compression Type	RAG	ICL	Synthetic	Summ.	Avg.
Fine-KV	59.9	**75.5**	54.1	**21.0**	52.6
+ SFT	**60.2**	73.3	**66.3**	**21.7**	**55.4**

Table 13: Performance of the compression model after SFT (compression ratio=4).

Methods	Peak Mem.	TTFT	TPOT	Latency
16K tokens input				
StreamingLLM	17.3	2.3	34.0	17.3
Knorm	17.3	2.3	34.3	17.3
SnapKV	17.3	2.4	32.0	18.3
Gist-based	16.9	2.2	35.5	16.6
32K tokens input				
StreamingLLM	19.6	4.1	36.0	18.3
Knorm	19.6	4.1	36.5	18.1
SnapKV	19.6	4.4	31.9	20.4
Gist-based	18.5	3.8	37.3	17.4
64K tokens input				
StreamingLLM	24.3	9.9	36.2	23.6
Knorm	24.3	9.9	35.6	24.2
SnapKV	24.3	10.1	36.7	24.1
Gist-based	21.5	7.9	35.5	22.3

Table 14: Quantitative efficiency comparison among popular token-eviction methods and gist-based methods. We report peak memory usage (GB), TTFT (s), TPOT (token/s), and Latency (s).

token-based architecture outperforms all token-eviction baseline methods across all sub-tasks. Further examinations reveal that the performance difference between gist-based methods and baselines is relatively small in tasks such as RAG and LongQA, but more significant in tasks like Synthetic Recall. This indicates that learnable gist token-based compression architectures have greater advantages in terms of compression quality and generalizability.

Efficiency We focus on four efficiency metrics commonly used in model inference: Peak Memory Usage, Time to First Token (TTFT), Time Per Output Token (TPOT), and Latency. For quantitative evaluation, we set up experiments with context input lengths of 16K, 32K, and 64K tokens, and an output length of 512 tokens. We run each method five times, averaging the results to obtain the final values.

Table 14 shows that the performance of different methods is similar across various efficiency metrics. At a 64K context length, the gist-based method demonstrates more significant advantages in speed and peak memory, which is mainly attributed to the sparse attention mechanism realized by segment-level progressive compression. In fact, methods such as StreamingLLM can also utilize chunked-prefill technology to achieve comparable, optimized peak memory usage. However, not all token-eviction strategies support chunked-prefill technology; for example, SnapKV relies on the output of the last window to determine which previous window should be discarded.

Methods	Train Time (Offline)	Train Time (Online)
Fine-KV	192.3(+0.0%)	192.3(+0.0%)
+AE	211.1(+9.8%)	211.1(+9.8%)
+TIE	193.6(+0.6%)	242.8(+26.3%)

Table 15: Training time (s) to complete one step (2M tokens) in different strategies.

Length	Model	CR.	RAG	ICL	Synthetic	Avg.
16K	Full	-	61.8	62.3	93.9	72.7
	Fine-KV	4	60.4	72.7	62.1	65.1
32K	Full	-	60.5	74.9	88.7	74.7
	Fine-KV	4	59.3	76.8	34.9	57.9

Table 16: Performance of compression models when inference length exceeds training length.

F Efficiency Analysis of Our Proposed Strategies

The strategies we propose effectively alleviate the flaws of existing gist token-based compression techniques. To further assess its efficiency and explore its applicability in large-scale continual pretraining, under the same settings as the main experiment in the paper based on LLAMA-3.1-8B (16K context, equivalent batch size = 2M tokens), we record the average time required for each mode to complete one training step.

As shown in Table 15, we observe that, in terms of training speed, the AE and TIE strategies only increase the time by 9.8% and 0.6%, respectively. Considering that token importance estimation can be largely pre-computed offline and independently, its overhead can be practically negligible. Even under the worst-case scenario of online estimation,

Ratio	Type	MMLU-Pro	BBH	GSM8K	HellaSwag
-	Full Attention	34.1	64.8	51.2	82.8
4	Coarse-Rec	34.1	53.8	50.3	81.9
	Coarse-KV	35.3	58.1	48.7	82.3
	Fine-KV	33.9	59.2	52.2	82.5
8	Coarse-Rec	34.1	54.6	51.9	82.0
	Coarse-KV	35.6	56.1	49.0	82.2
	Fine-KV	34.6	56.8	51.9	82.5
16	Coarse-Rec	34.1	53.2	50.0	81.9
	Coarse-KV	35.6	55.7	50.1	82.2
	Fine-KV	34.3	56.0	51.7	82.2
32	Coarse-Rec	34.1	54.8	50.8	81.9
	Coarse-KV	35.6	50.6	50.5	82.2
	Fine-KV	33.6	55.0	50.6	82.2

Table 17: Performance on weak context-dependent tasks.

TIE only adds 26.3% time overhead. This demonstrates that the proposed methods are cost-effective and have the potential for large-scale scalability.

G Extrapolation Capabilities

This work explores a segment-wise context compression method that can effectively reduce the maximum length that each transformer block needs to model. For example, taking LLAMA3-8B as an example, assuming a fixed compression ratio of 4 and a segment length of 1K, the context length after continue-training would be the same as the pre-training length, which is 8K. Even if the user's input context length reaches 16K, exceeding the maximum length after continue-training, the actual maximum length that each transformer block needs to model would only be (16K-1K)/4+1K=4.75K, which still falls within the pre-trained context length of the model. Since the model has already learned the corresponding positional encodings during pre-training, this method holds promise for extrapolating actual inference lengths.

Using LLAMA3.1-8B as the base model, we evaluate the compressed model trained with 16K contexts on tasks involving 32K contexts. As shown in Table 16, the results indicate that the compressed model continues to perform well even with context lengths multiple times longer than the training length. This suggests that the ability to read context from gist tokens is generalizable.

H Results with More Compression Ratios

Constrained by space, results for the 32:1 compression ratio are not presented in the main text. Therefore, a comprehensive set of results is available in Table 17, Table 19, and Table 20.

Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
StreamingLLM	48.1	0.2	15.6	10.9	22.8	13.2	23.2	16.9
Knorm	47.9	0.1	23.5	6.6	5.3	11.9	20.7	12.1
SnapKV	50.2	3.2	26.0	8.5	5.9	10.2	20.6	13.1
AdaKV	52.8	2.5	24.8	26.8	17.0	10.9	20.5	21.4
Gist-based (Fine-KV)	**60.6**	**23.4**	**40.3**	**70.6**	**40.6**	**21.0**	**63.0**	**46.2**

Table 18: Long context performance comparison with popular token-eviction methods

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	61.8	39.9	41.6	62.3	93.9	23.8	66.1	55.6
	Full Attention, Finetune	61.7	38.5	42.3	60.0	91.0	24.1	65.7	54.7
4	Coarse-grained, Recurrent	49.9	2.1	35.2	29.4	11.2	18.2	59.3	29.3
	Coarse-grained, KV Cache	51.7	5.2	33.9	36.0	14.2	17.6	57.8	30.9
	Fine-grained, KV Cache	**60.6**	**23.4**	**40.3**	**70.6**	**40.6**	**21.0**	**63.0**	**46.2**
8	Coarse-grained, Recurrent	49.8	1.3	36.0	25.9	11.2	**17.7**	58.6	28.6
	Coarse-grained, KV Cache	50.8	3.8	36.5	33.6	13.5	16.1	57.2	30.2
	Fine-grained, KV Cache	**57.6**	**14.5**	**40.2**	**68.1**	**26.9**	16.7	**60.7**	**40.7**
16	Coarse-grained, Recurrent	49.9	1.4	34.9	20.8	11.2	**17.8**	57.5	27.6
	Coarse-grained, KV Cache	50.2	4.4	34.2	29.1	13.1	16.7	58.1	29.4
	Fine-grained, KV Cache	**55.4**	**10.0**	**40.4**	**49.3**	**13.8**	**16.3**	**59.2**	**34.9**
32	Coarse-grained, Recurrent	49.3	1.2	33.6	21.1	11.1	**17.5**	58.2	27.4
	Coarse-grained, KV Cache	49.9	2.6	34.2	25.0	**12.2**	17.1	58.2	28.5
	Fine-grained, KV Cache	**53.1**	**3.1**	**37.6**	**36.4**	11.9	16.1	**59.2**	**31.0**

Table 19: Performance comparison among full attention and compression architectures on long context tasks. **Bold** indicates the best result along the same compression ratio.

Ratio	Compression Type	RAG	Rerank	LongQA	ICL	Synthetic	Summ.	Code	Average
-	Full Attention	61.8	39.9	41.6	62.3	93.9	23.8	66.1	55.6
4	Fine-grained, KV Cache	60.6 (+0.0)	23.4 (+0.0)	40.3 (+0.0)	70.6 (+0.0)	40.6 (+0.0)	21.0 (+0.0)	62.0 (+0.0)	46.1 (+0.0)
	+ Fine-grained AE	60.9 (+0.3)	27.4 (+4.0)	40.8 (+0.5)	72.0 (+1.4)	62.0 (+21.4)	22.3 (+1.3)	62.9 (+0.9)	49.8 (+3.7)
	+ Segment-wise TIE	60.4 (-0.2)	27.0 (+3.6)	41.2 (+0.9)	72.7 (+2.1)	54.3 (+13.7)	20.2 (-0.8)	62.1 (+0.1)	48.3 (+2.2)
	+ Both Strategies	61.1 (+0.5)	27.4 (+4.0)	40.3 (+0.0)	75.0 (+4.4)	62.1 (+21.5)	22.2 (+1.2)	62.9 (+0.9)	**50.1** (+4.0)
8	Fine-grained, KV Cache	57.6 (+0.0)	14.5 (+0.0)	40.2 (+0.0)	68.1 (+0.0)	26.9 (+0.0)	16.7 (+0.0)	60.7 (+0.0)	40.7 (+0.0)
	+ Fine-grained AE	58.3 (+0.7)	15.6 (+0.9)	39.8 (-0.4)	68.7 (+0.6)	34.8 (+7.9)	18.5 (+1.8)	61.3 (+0.6)	42.4 (+1.7)
	+ Segment-wise TIE	58.1 (+0.4)	17.6 (+3.1)	40.0 (-0.2)	70.0 (+1.9)	30.2 (+3.3)	17.7 (+1.0)	60.7 (+0.0)	42.0 (+1.3)
	+ Both Strategies	58.3 (+0.7)	19.7 (+5.2)	40.4 (+0.0)	70.7 (+2.6)	35.2 (+8.9)	19.5 (+2.8)	61.4 (+0.7)	**43.6** (+2.9)
16	Fine-grained, KV Cache	55.4 (+0.0)	10.0 (+0.0)	40.4 (+0.0)	49.3 (+0.0)	13.8 (+0.0)	16.3 (+0.0)	59.2 (+0.0)	34.9 (+0.0)
	+ Fine-grained AE	55.6 (+0.2)	11.3 (+1.3)	40.4 (+0.0)	47.1 (+0.3)	14.7 (+0.9)	16.2 (-0.1)	59.6 (+0.4)	35.0 (+0.1)
	+ Segment-wise TIE	55.6 (+0.2)	10.4 (+0.4)	40.7 (+0.3)	55.5 (+8.4)	14.8 (+1.0)	15.3 (-1.0)	58.1 (-1.1)	35.7 (+0.8)
	+ Both Strategies	56.3 (+0.9)	12.7 (+2.7)	41.7 (+1.3)	56.3 (+7.0)	14.9 (+1.1)	15.7 (-0.6)	59.6 (+0.4)	**36.7** (+1.8)
32	Fine-grained, KV Cache	53.1 (+0.0)	3.1 (+0.0)	37.6 (+0.0)	36.4 (+0.0)	11.9 (+0.0)	16.1 (+0.0)	59.2 (+0.0)	31.0 (+0.0)
	+ Fine-grained AE	54.3 (+1.2)	4.6 (+1.5)	39.3 (+1.7)	34.1 (-2.3)	13.1 (+1.2)	17.1 (+1.0)	59.8 (+0.6)	31.8 (+0.8)
	+ Segment-wise TIE	53.1 (+0.0)	4.6 (+1.5)	40.3 (+2.7)	43.6 (+7.2)	13.1 (+1.2)	17.0 (+0.9)	59.8 (+0.6)	**33.1** (+2.1)
	+ Both Strategies	54.4 (+1.3)	4.9 (+1.8)	39.8 (+2.2)	41.8 (+5.4)	13.1 (+0.9)	17.1 (+1.0)	59.8 (+0.6)	33.0 (+2.0)

Table 20: Performance comparisons using our methods, with the best "average" results bolded for clarity.

A Synthetic Example in PopQA	
Subject is relevant, and needle type is food	
Subject:	John Peter Jukes
Document 1:	For the cartoonist with the same name see John Jukes. The Right Reverend John Peter Jukes (7 August 1923) was an English prelate of the Roman Catholic Church. He was a member of the Conventual Franciscans. Jukes was born in Eltham...
Document 2:	Richard Jukes was born on 9 October 1804 at Goathill, and died 10 August 1869. He served as a Primitive Methodist minister from 1827 to 1859. Jukes married Phoebe Pardoe in 1825, and later, widowed, he married Charlotte...
Golden doc:	[Some content] John Peter Jukes's special food is beef burger. [The rest of content...]
More documents:	...
Question:	What's the special food of John Peter Jukes?
Subject is relevant, and needle type is number	
Subject:	John Peter Jukes
Document 1:	For the cartoonist with the same name see John Jukes. The Right Reverend John Peter Jukes (7 August 1923) was an English prelate of the Roman Catholic Church. He was a member of the Conventual Franciscans. Jukes was born in Eltham...
Document 2:	Richard Jukes was born on 9 October 1804 at Goathill, and died 10 August 1869. He served as a Primitive Methodist minister from 1827 to 1859. Jukes married Phoebe Pardoe in 1825, and later, widowed, he married Charlotte...
Golden doc:	[Some content] John Peter Jukes's special number is 51681396. [The rest of content...]
More documents:	...
Question:	What's the special number of John Peter Jukes?
Subject is irrelevant, and needle type is food	
Subject:	John Peter Jukes
Document 1:	For the cartoonist with the same name see John Jukes. The Right Reverend John Peter Jukes (7 August 1923) was an English prelate of the Roman Catholic Church. He was a member of the Conventual Franciscans. Jukes was born in Eltham...
Document 2:	Richard Jukes was born on 9 October 1804 at Goathill, and died 10 August 1869. He served as a Primitive Methodist minister from 1827 to 1859. Jukes married Phoebe Pardoe in 1825, and later, widowed, he married Charlotte...
Golden doc:	[Some content] Mr. Tree's special food is beef burger. [The rest of content...]
More documents:	...
Question:	What's the special food of Mr. Tree?
Subject is irrelevant, and needle type is number	
Subject:	John Peter Jukes
Document 1:	For the cartoonist with the same name see John Jukes. The Right Reverend John Peter Jukes (7 August 1923) was an English prelate of the Roman Catholic Church. He was a member of the Conventual Franciscans. Jukes was born in Eltham...
Document 2:	Richard Jukes was born on 9 October 1804 at Goathill, and died 10 August 1869. He served as a Primitive Methodist minister from 1827 to 1859. Jukes married Phoebe Pardoe in 1825, and later, widowed, he married Charlotte...
Golden doc:	[Some content] Mr. Tree's special number is 51681396. [The rest of content...]
More documents:	...
Question:	What's the special number of Mr. Tree?

Table 21: A synthetic example in PopQA for evaluate "Lost if surprise". The Red parts denote synthetic needles inserted to the dataset.