KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches

Jiayi Yuan[*1], **Hongyi Liu**[*1], **Shaochen (Henry) Zhong**[*1], **Yu-Neng Chuang**[1], **Songchen Li**[1], **Guanchu Wang**[1], **Duy Le**[1,3], **Hongye Jin**[2], **Vipin Chaudhary**[3], **Zhaozhuo Xu**[4], **Zirui Liu**[♣5], **Xia Hu**[1]

[1]Rice University, [2]Texas A&M University, [3]Case Western Reserve University, [4]Stevens Institute of Technology, [5]University of Minnesota Twin Cities

Abstract

Long context capability is a crucial competency for large language models (LLMs) as it mitigates the human struggle to digest long-form texts. This capability enables complex task-solving scenarios such as book summarization, code assistance, and many more tasks that are traditionally manpower-intensive. However, transformer-based LLMs face significant challenges with long context input due to the growing size of the KV cache and the intrinsic complexity of attending to extended inputs; where multiple schools of efficiency-driven approaches — such as KV cache quantization, token dropping, prompt compression, linear-time sequence models, and hybrid architectures — have been proposed to produce efficient yet long context-capable models. Despite these advancements, no existing work has comprehensively benchmarked these methods in a reasonably aligned environment. In this work, we fill this gap by providing a taxonomy of current methods and evaluating 10+ state-of-the-art approaches across seven categories of long context tasks. Our work reveals numerous previously unknown phenomena and offers insights — as well as a friendly workbench — for the future development of long context-capable LLMs. The source code is available at https://github.com/henryzhongsc/longctx_bench.

1 Introduction

Large Language Models (LLMs) have gained significant popularity and recognition due to their exceptional generalizability across a wide range of intellectual tasks. Like any other tool, their most precious utility is demonstrated when they enable us to accomplish tasks beyond our innate capabilities (Brown et al., 2020; Taylor et al., 2022; Yuan et al., 2023). For instance, while driving nails with bare hands is impractical, a hammer makes it feasible. Similarly, humans struggle with digesting and retaining long information, making it essential for LLMs to bridge this gap. The need for long-context capable LLMs is almost universally agreed upon, with different LLM service providers racing to launch models with even greater context lengths. For example, Google's Gemini 1.5 supports a context length of 128K tokens (Reid et al., 2024), and Anthropic's Claude 3 offers a context length of 200K tokens.

However, this powerful long context capability comes with significantly higher costs. In long context scenarios, the key-value cache (KV cache) — which stores attention keys and values during generation to prevent re-computation — becomes the new memory and speed bottlenecks, as its size grows linearly with the number of tokens in the batch. For instance, a 500B model with a batch size of 128 and a context length of 8,192 typically requires a 3TB KV cache, imposing a substantial processing burden even on the most advanced hardware solutions (Pope et al., 2023). Similarly, in open-source models like QWen (Bai et al., 2023a), the KV cache size for a 4K context is 0.91 GB, whereas, for a 100K context, it is 22.8 GB (Fu, 2024) — which is a non-negligible growth regardless of the serving scenario. Given the limited memory space available for serving the model, supporting longer contexts usually requires reducing the number of requests that can be processed, leading to higher inference costs.

Naturally, many efficiency-driven approaches have been proposed to enable LLMs to handle long contexts with reduced resource burdens, with a healthy selection of them featured in Table 1. These approaches range from quantizing the KV cache into lower precision formats (Sheng et al., 2023; Zhao et al., 2024; Liu et al., 2024b), evicting unimportant tokens to maintain a constant KV cache size (Xiao et al., 2023; Zhang et al., 2024d), compress-

[*]Equal contribution, order determined by rolling dices.
[♣]Project lead and correspond to Shaochen (Henry) Zhong <henry.zhong@rice.edu> and Zirui Liu <zrliu@umn.edu>.

Table 1: Featured methods in our benchmark. "KV Cache Complexity" is the complexity w.r.t. the number of input tokens. "Sys. Supports" refers to the availability of custom CUDA kernels to support fast serving. "N/A" means it can be directly accelerated by existing infrastructure. We note that the "No" system support for H_2O (Zhang et al., 2024d) means it lacks the FlashAttention (Dao et al., 2022) compatible CUDA kernels, making it unsuitable for direct use in an online setting. However, it still offers performance benefits when used in the off-load setting.

Method	Taxonomy	KV Cache Complexity	Sys. Supports?
Mamba (Gu and Dao, 2023)	Linear-time Model	KV cache free	Yes
Mamba 2 (Dao and Gu, 2024)			Yes
RWKV (Peng et al., 2023)			Yes
RecurrentGemma (Botev et al., 2024)	Linear-time Model + Local Attention	Constant	Yes
StreamingLLM (Xiao et al., 2023)	Token Dropping	Constant	Yes
H_2O (Zhang et al., 2024d)			No
InfLLM (Xiao et al., 2024)			Yes
LLMLingua2 (Pan et al., 2024)	Prompt Compression	Constant	N/A
FlexGen (Sheng et al., 2023)	Quantization	Linear	Yes
KIVI (Liu et al., 2024b)			Yes

ing long prompt into a shorter input (Jiang et al., 2023b; Pan et al., 2024; Chuang et al., 2024), or exploring KV cache-free architectural designs (Gu and Dao, 2023; Peng et al., 2023; Yang et al., 2023; Qin et al., 2024) and its hybrids with transformers (De et al., 2024; Lieber et al., 2024). However, to the best of our knowledge, **no prior art has provided a comprehensive benchmark to analyze the performance retention of different long context-capable compression methods**[1] (which is also non-trivial to setup; more on this in Section 3.2). To fill this gap, we aim to answer the following question:

How do different long context-capable approaches perform under different long context tasks?

This benchmark offers an accessible and reproducible pipeline to evaluate a diverse set of modern long-context compression methods from various schools of thought. It assesses these methods against multiple tasks requiring different long-context capabilities. Our main contributions are summarized as follows:

- **Comprehensive benchmarking, detailed analysis, and actionable insights:** We provide a comprehensive evaluation report that covers 10+ long context-capable efficient approaches under 65 different settings, against 7 categories of long context tasks (Mohtashami and Jaggi, 2023; Reid et al., 2024; Bai et al., 2023b). We then walk through how to digest such mass results and provide analyses and discussion upon many previously unknown phenomena. Finally, we offer several actionable insights for future research advancement.
- **Minimalistic, reproducible, yet extensible platform:** Given the non-trivial effort to set up the evaluation pipeline, we open source our benchmark implementations for future scholars. We intensionally make our code base in a minimalistic fashion for easier hacking and reproducing needs, yet we keep it extensible to include alternative or future-coming approaches that are not under in our already extensive, but certainly not exhaustive, benchmark coverage.

2 Reviewing Different Schools of Efficient Long Context Handling Approcahes

Before going into the details of the experiment, we will briefly introduce different schools of long context-capable approaches and their corresponding exemplary methods. In Table 1, we present a comprehensive overview of the school of long context optimization methods, including their KV cache complexities and the current support for system-level optimization. RNN-based models do not have a KV cache. Mixed models, token dropping methods, and prompt compression methods have fixed-size KV caches, which are independently configured by each method. Quantization methods compress the KV cache by a proportion; thus, the KV cache complexity still increases lin-

[1] Due to the lack of directly related work, we provide a brief walkthrough of loosely related arts — which are often long context datasets evaluated on vanilla baseline models with limited focus on compression methods — in Appendix C.

early with sequence length. Regarding system support scenarios, to the best of our knowledge, most methods have varying levels of system-level optimization, whereas some token-dropping methods are still under-optimized. More on this in Section 4.

2.1 Linear-Time Sequence Models and Mixed Architecture

There is a growing body of recent works that have developed linear-time sequence models, such as Mamba (Gu and Dao, 2023), Mamba2 (Dao and Gu, 2024), RWKV (Peng et al., 2023), HGRN (Qin et al., 2024), MEGA (Ma et al., 2022), GLA (Yang et al., 2023), and RetNet (Sun et al.). The fundamental difference between linear-time sequence models and transformers lies in how they handle context. Linear-time sequence models compress the context into a smaller state, whereas transformers store the entire context within attention mechanisms. During the auto-regressive inference, every time the model generates a new token, transformers will "review" all previous tokens by explicitly storing the entire context (i.e., KV cache). In contrast, there is no "reviewing" mechanism in linear-time sequence models, as they explicitly mix the input tokens into finite states.

From the above analysis, it is expected that pure linear-time sequence models are not well-suited for retrieval-related tasks, as they mix key information with other tokens. Thus, another line of work is to combine the linear-time sequence models and transformers. For example, Griffin (De et al., 2024) and RecurrentGemma (Botev et al., 2024) combine input-dependent RNNs with local attention; and Jamba (Lieber et al., 2024) combines full attention layers and Mamba layers.

2.2 Quantization

A simple yet effective approach to reducing the size of KV cache to enable a larger context is to quantize the floating-point numbers (FPN) in the KV cache using fewer bits. Specifically, the B-bit integer quantization-dequantization process can be expressed as:

$$Q(\mathbf{X}) = \lfloor \frac{\mathbf{X} - z_X}{s_X} \rceil, \quad \mathbf{X}' = Q(\mathbf{X}) \cdot s_X + z_X,$$

where $z_X = \min \mathbf{X}$ is the zero-point, $s_X = (\max \mathbf{X} - \min \mathbf{X})/(2^B - 1)$ is the scaling factor, and $\lfloor \cdot \rceil$ is the rounding operation.

FlexGen (Sheng et al., 2023) utilized group-wise quantization, achieving 4bit quantization compared to standard 16bit with minimal accuracy loss. Following this, several other quantization methods have been proposed specifically for the KV cache (Zhao et al., 2024; Yang et al., 2024; Dong et al., 2024). Recently, KIVI (Liu et al., 2024b) and KVQuant (Hooper et al., 2024) advanced KV cache quantization to even lower bits by introducing per-channel quantization, which involves grouping tensor elements along the channel dimension, based on the discovery of channel outliers in the key cache. Following this finding, some other works continue to optimize this process (Kang et al., 2024; Duanmu et al., 2024; Zandieh et al., 2024). Furthermore, based on these findings, the latest research has pushed quantization to 1bit (Zhang et al., 2024b). The transformer-based LLM inference workflow involves two stages: i) *prefill stage*, where the input prompt is used to generate KV cache and the first output token; and ii) *decoding stage*, where the model uses and updates KV cache to generate the next token one by one. **We emphasize that for all KV cache quantization methods evaluated in this paper, the quantized KV cache is not used in prefill time**. That means that KV cache quantization only affects the decoding phase.

2.3 Token Dropping

Based on the observation that attention scores are highly sparse, token dropping-based methods drop the unimportant token — or similar attention components — from the KV cache (Zhang et al., 2024d; Xiao et al., 2023, 2024; Li et al., 2024a,c; Liu et al., 2024a; Ge et al., 2023; Jiang et al., 2024). **Token dropping-based methods fall into two main categories: dropping tokens during prefill or dropping tokens after prefill.** Dropping tokens during prefill means that tokens are dropped while generating the KV cache. In contrast, dropping tokens after prefill means generating the full KV cache first, then removing the unimportant tokens from it. Given transformers inference process typically involves two phases, i.e., prefill and decoding, **while dropping tokens during prefill can typically enable longer sequence length and faster prefill speed, we note that dropping tokens after prefill consistently yields better results across various settings**. This is because many token-dropping methods rely on accurate attention scores to determine token importance, which benefits from generating the full KV cache first. In our benchmark, methods that drop tokens during prefill include StreamingLLM (Xiao et al., 2023) and InfLLM (Xiao et al., 2024), where H_2O (Zhang et al.,

2024d) represents methods that drop tokens after prefill. We closely follow the official or endorsed implementation of each method, with more details shared in Appendix B.3.

2.4 Prompt Compression

Soft Prompt Compression Most existing work focuses on converting lengthy prompts into trainable soft prompts optimized with specific LLMs. One approach uses knowledge distillation to transform hard prompts into soft prompts (Wingate et al., 2022). Another leverages LLM summarization to condense prompts by segmenting and compressing information (Chevalier et al., 2023). Gist Token (Mu et al., 2023) creates customized prefix soft prompts via a virtual soft prompt predictor. However, these methods are often model-or-even-task-specific, requiring training tailored to specific LLMs, and therefore come with limited adaptability. In this work, we focus on general compression methods for fair comparison with other KV cache compression approaches.

Natural Language Prompt Compression Methods like LLMLingua family (Pan et al., 2024; Jiang et al., 2023b) enhance LLM performance on long-context tasks by converting long prompts into short prompts while maintaining their natural language format, and thus naturally adaptable (and often even transferable) to all LLMs. LLMLingua employs a budget controller to dynamically allocate compression ratios to different prompt parts, ensuring semantic integrity. Unlike LLMLingua's general approach, some hard prompt compression methods, like Nano-Capsulator (Chuang et al., 2024), provide task-specific compression to preserve long prompt performance and are therefore excluded in our benchmark.

2.5 Other Schools of Thought: Linear Attention, Merging, and More.

Other than the above-featured approaches, several notable avenues for efficient long context handling include *linear attention* and *merging*. Linear Attention is a well-explored area of transformer modification with many impactful prior arts like LinearAttention (Katharopoulos et al., 2020), MetaFormer (Yu et al., 2022), LinFormer (Wang et al., 2020) and more, with most of them mainly focus on vision or natural language understanding tasks. To the best of our knowledge, Infini-Attention by Munkhdalai et al. (2024) is likely one of the most impactful linear attention approaches under the LLM context.

KV cache merging is also a popular approach due to the mainstream adaptation of GQA (Ainslie et al., 2023), GQA and MQA (Shazeer, 2019) conduct merging at the transformer head dimension to enable KV cache reuse. Similar cache-sharing strategies have been developed at the layer or token levels (Sun et al., 2024; Brandon et al., 2024; Wu and Tu, 2024; Nawrot et al., 2024). Most of the techniques proposed under this category require intervention during the pre-training stage.

Unfortunately, we are unable to feature these schools of thought, since our work requires scaled-up open-source models in such designs to be available in the first place. With the lack of such availability, we cannot feature them *per se* in our evaluation. However, we are able to feature Mamba 2 (Dao and Gu, 2024) — a model family with a generalized linear attention mechanism — at the 2.7B scale and thus provide some relevant results. We also direct our readers' attention to some recent attention variants like MLA (DeepSeek-AI, 2024).

3 Benchmarking

Benchmarking such a variety of methods in a reasonable manner requires significant effort in terms of experiment design, execution, and computational resources. We first introduce the datasets and methods covered, along with the justifications for their selection. Then, we detail the experiment setup and explain how to interpret our experiment reports. Finally, we analyze the reported results by highlighting some interesting phenomena and providing insights for future scholars. All experiments are conducted on one or more 80G NVIDIA A100 GPUs under DGXA100 servers.

3.1 Coverage

Tasks and Models. We focus on **16 different long context tasks under 7 major categories**, each requiring different long context handling abilities and covering key application scenarios. We provide a brief walkthrough of each task category as follows: (1) *Single-doc QA*, which tests the long context understanding ability with longer documents. (2) *Multi-Doc QA*, which needs to extract and combine information from several documents to obtain the answer; (3) *Summarization*, which requires a global understanding of the whole context; (4) *Few-shot Learning*, which is a practical setting requiring long-context understanding over provided examples; (5) *Synthetic Task*, which is

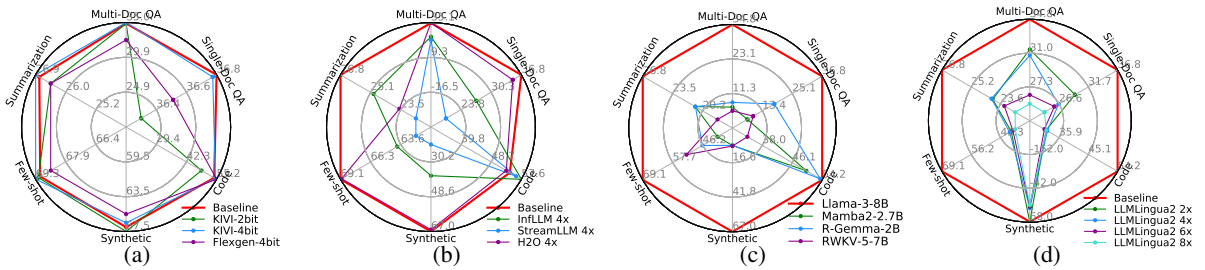
Table 2: Performance of KV cache quantization, token dropping, prompt compression, RNNs, and hybrid methods on our benchmark. For methods with residual full precision inputs like KIVI, we calculate the "Comp. Ratio" against 10k input length. "LB Avg." refers to the average results on LongBench. Results are abbreviated; please refer to Appendix D for our full report.

Model	Method	Comp. Ratio	Single. QA	Multi. QA	Summ.	Few-shot	Synthetic	Code	**LB Avg.**	**Needle**
Meta-Llama-3-8B-Instruct	Baseline	1.00×	36.8	34.8	26.8	69.1	67.0	54.2	45.2	100.0
	KIVI-2bit	5.05×	36.2	34.8	26.4	69.2	67.5	48.8	44.3	100.0
	KIVI-4bit	3.11×	36.8	35.0	26.9	69.3	66.5	54.7	45.3	100.0
	FlexGen-4bit	3.20×	36.5	32.4	26.4	68.6	65.5	55.2	44.5	100.0
	InfLLM-2x	2.00×	31.8	30.8	25.7	67.6	57.5	55.8	42.5	22.7
	InfLLM-4x	4.00×	27.1	24.7	25.0	63.9	37.5	57.6	38.3	20.7
	InfLLM-6x	6.00×	24.4	23.4	24.3	61.1	29.5	59.2	36.5	24.7
	InfLLM-8x	8.00×	21.0	21.0	23.7	60.3	18.0	59.9	34.4	22.0
	StreamLLM-2x	2.00×	26.1	28.8	24.6	66.5	34.0	55.6	38.9	29.0
	StreamLLM-4x	4.00×	20.5	22.2	22.7	62.2	21.0	56.1	34.4	22.3
	StreamLLM-6x	6.00×	17.4	18.7	21.4	60.1	14.5	59.0	32.3	18.0
	StreamLLM-8x	8.00×	15.7	18.0	20.5	55.9	8.0	58.1	30.3	18.0
	H_2O-2x	2.00×	35.8	34.8	25.4	69.1	66.0	54.4	44.7	100.0
	H_2O-4x	4.00×	35.0	35.1	23.6	69.0	66.0	53.2	44.0	100.0
	H_2O-6x	6.00×	33.9	35.1	22.7	69.1	66.0	53.1	43.6	100.0
	H_2O-8x	8.00×	33.7	35.0	22.2	69.1	65.5	52.7	43.4	100.0
	LLMLingua2-2x	2.00×	29.4	31.5	24.1	38.6	68.0	31.9	33.5	51.3
	LLMLingua2-4x	4.00×	26.5	30.8	24.1	39.3	22.5	32.2	29.9	8.3
	LLMLingua2-6x	6.00×	25.8	26.4	23.4	37.9	18.0	31.3	28.1	0.7
	LLMLingua2-8x	8.00×	24.0	25.4	22.9	36.9	13.0	31.9	26.9	0.0
Mistral-7B-Instruct-v0.2	Baseline	1.00×	32.5	25.8	27.9	66.7	89.3	54.0	43.8	99.0
	KIVI-2bit	5.05×	31.3	24.7	27.6	66.8	80.8	53.7	42.6	99.0
	KIVI-4bit	3.11×	32.3	25.8	27.9	66.9	89.4	54.0	43.8	99.0
	FlexGen-4bit	3.20×	33.0	24.4	27.8	66.2	83.0	53.7	43.0	98.3
	InfLLM-2x	2.00×	30.7	24.8	26.8	65.1	65.8	54.2	41.1	64.3
	InfLLM-4x	4.00×	25.4	23.7	25.5	63.4	41.4	54.0	37.5	29.7
	InfLLM-6x	6.00×	23.8	21.0	25.0	61.6	32.6	53.4	35.6	32.3
	InfLLM-8x	8.00×	22.2	19.6	24.3	62.0	26.2	53.8	34.5	27.0
	StreamLLM-2x	2.00×	24.6	22.0	25.3	64.5	47.1	53.0	37.5	54.7
	StreamLLM-4x	4.00×	20.1	19.9	23.3	61.3	31.6	53.9	34.2	32.0
	StreamLLM-6x	6.00×	18.2	16.0	22.1	59.6	25.3	54.9	32.2	25.0
	StreamLLM-8x	8.00×	17.0	15.2	21.4	58.3	16.9	54.9	30.8	19.3
	H_2O-2x	2.00×	31.9	25.4	26.8	66.8	87.7	53.8	43.2	97.3
	H_2O-4x	4.00×	30.4	23.9	25.1	67.2	82.9	53.1	41.9	93.3
	H_2O-6x	6.00×	29.0	22.8	24.3	66.9	82.0	52.5	41.1	85.7
	H_2O-8x	8.00×	27.8	21.8	23.9	67.0	79.5	52.3	40.4	80.0
	LLMLingua2-2x	2.00×	28.6	23.0	26.4	45.6	54.9	31.7	32.6	41.7
	LLMLingua2-4x	4.00×	25.0	21.3	24.6	39.2	14.0	33.1	27.4	9.7
	LLMLingua2-6x	6.00×	21.2	17.4	23.3	38.9	8.9	34.7	25.4	0.0
	LLMLingua2-8x	8.00×	19.6	16.0	22.9	38.5	8.0	35.5	24.7	0.0
Mamba	Mamba-2.8B	-	7.3	6.3	19.1	39.0	1.2	47.6	20.8	10.7
	Mamba-Chat-2.8B	-	9.2	6.9	21.2	37.5	3.7	47.7	21.6	10.7
	Mamba2-2.7B	-	7.5	6.7	21.0	40.5	4.1	49.9	22.1	9.0
RWKV	RWKV-5-World-7B	-	9.8	5.4	18.5	52.4	4.5	34.0	22.1	3.7
R-Gemma	R-Gemma-2B-it	-	18.1	8.3	20.9	46.3	4.0	53.7	26.1	23.3
	R-Gemma-9B-it	-	24.5	21.9	21.9	54.5	9.0	60.8	33.2	26.7

designed to test the model's ability on specific scenarios and patterns; (6) *Code Completion*, which is designed to test the model's long-context ability in code auto-completion tasks; (7) *Needle-in-a-Haystack Test*, which involves finding specific information within a large volume of text.

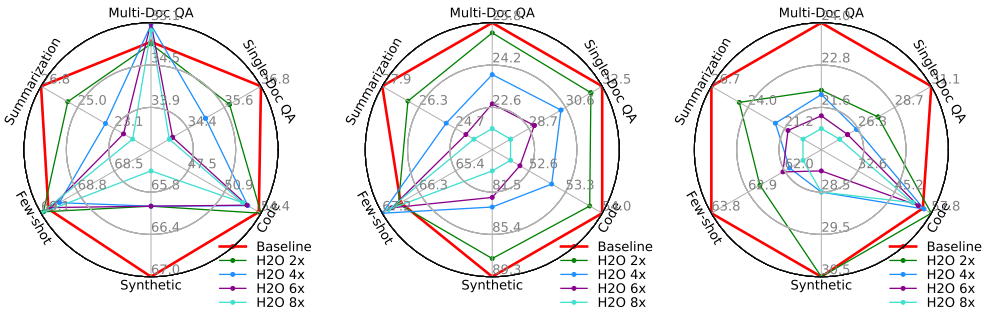
For categories (1)-(6), we directly adopt them from the LongBench dataset (Bai et al., 2023b). For the (7) Needle-in-a-Haystack Test, we largely follow the format of the original passkey retrieval task (Mohtashami and Jaggi, 2023) while including some modern modifications set forward by Arize-ai and the technical report of Gemini 1.5 (Reid et al., 2024). We refer our readers to Appendix A for further details.

For models, we elect to cover **3 representative transformer-based LLMs and 3 pure or hybrid linear-time sequence model families**. For transformer-based LLMs, we opt for Mistral-7b-Instruct-v0.2 (Jiang et al., 2023a), Longchat-



(a) (b) (c) (d)

Figure 1: The rador plot of different methods (a) Llama-3-8B Llama-3-8B w./ Quant. (b) Llama-3-8B w./ Token Dropping (c) Linear-time sequence models and mixed Architecture (d) Llama-3-8B w./ Prompt compression.



(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

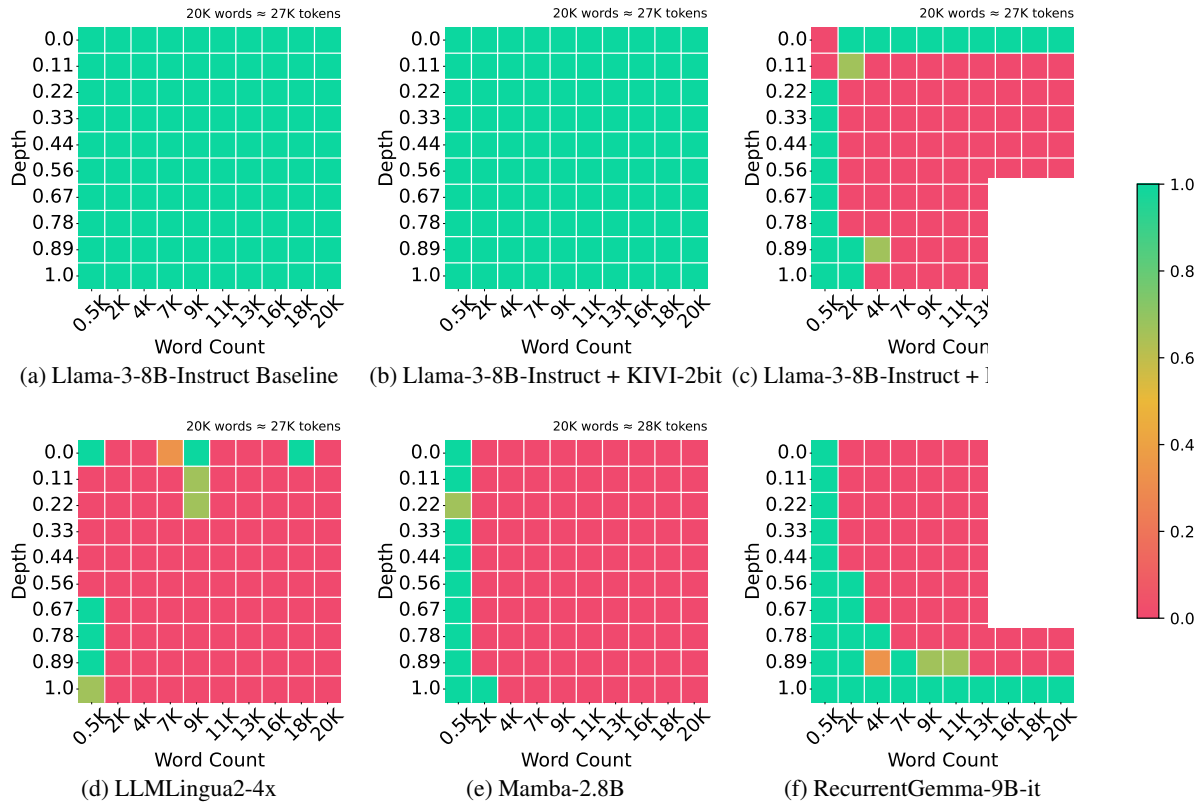
Figure 2: H_2O with different compression ratios on three commonly used LLMs.

7B-v1.5-32k (Li et al., 2023a) and Llama-3-8B-Instruct (AI@Meta, 2024) to provide a coverage of SOTA long-context capable model as well as the most recent progress of open-source LLMs. For linear-time sequence models and their hybrids, we evaluated Mamba-2.8B (Gu and Dao, 2023), Mamba2-2.7b (Dao and Gu, 2024), Mamba-Chat-2.8B (Mattern and Hohr, 2023), RWKV-5-World (Peng et al., 2023), and RecurrentGemma-2b/9b-Instruct (Botev et al., 2024). We refer readers to Appendix B for more model-related details.

Methods and Hyperparameter Settings. As shown in Table 1, we select representative methods ranging from KV cache-free to linear complexity KV cache. Apart from the linear-time sequence models and their hybrids introduced above, we opt for the following compression methods: For *quantization*, we adopt KIVI (Liu et al., 2024b), INT4 per-token quantization in FlexGen (Sheng et al., 2023); For *Token dropping*, we adopt StreamingLLM (Xiao et al., 2023), H_2O (Zhang et al., 2024d), and InfLLM (Xiao et al., 2024). For *Prompt Compression*, we adopt LLMLingua2 (Pan et al., 2024). We note that **although token dropping-based methods are usually designed with a constant KV cache size in mind, we modify them to adapt linear compression schemes for fair comparison with other methods.** We share more method-specific details in Appendix B.3.

3.2 Experiment Setup and Report Digestion

Given the vastly different design principles employed in different schools of long context handling methods, it is, in fact, impossible to achieve a global alignment where all covered methods are considered fairly aligned against each other. For example, while KV cache quantization methods like FlexGen (Sheng et al., 2023) can adapt to different data precision, they can never be aligned with any KV cache-free approaches like Mamba (Gu and Dao, 2023). Similarly, token dropping approaches typically employ a constant size of kept tokens and evict everything else, making their compression gain dynamic against inputs of different lengths; and, again, not alignable with KV cache quantization methods nor KV cache-free approaches. Note that the abovementioned issues are merely some alignment hardships due to conflicts in different long contexts when handling schools. In reality, two long context-specific methods — even under the same school — can also bring further complications: e.g., KIVI (Liu et al., 2024b) includes a full precision sliding window for the most recent tokens, while FlexGen (Sheng et al., 2023) doesn't. Further, known that models like Mamba (Gu and Dao, 2023) and RWKV (Peng et al., 2023) are typically pre-trained on open-source datasets, their architecture potentials cannot be fairly evaluated compared to models like Llama-3 — which are pretrained upon proprietary data corpus and done



(a) Llama-3-8B-Instruct Baseline (b) Llama-3-8B-Instruct + KIVI-2bit (c) Llama-3-8B-Instruct +

(d) LLMLingua2-4x (e) Mamba-2.8B (f) RecurrentGemma-9B-it

Figure 3: Needle-in-a-Haystack results on Llama-3-8B-Instruct, linear-time sequence models, and mixed architectures. The best method in each school of approaches is featured with comparable compression ratios. The same length of input might convert to different numbers of tokens per different models, as noted in the upper right corners.

so with an overtrained recipe that has proven to be beneficiary. More on this in Appendix E.

As the best alternative, we opt to compress different methods towards a range of available target compression ratios shown in Table 2. For KV cache quantization methods, we derive such compression ratios by referring to the reduction in KV cache memory size against full precision KV cache. For token dropping approaches, we forgo their typical constant kept token setup and dynamically adjust the amount of evicted tokens upon the length of each input request. For hard prompt compression, we simply compress the final hard prompt to or below the target compression ratio. We keep KV cache-free methods in their vanilla forms as they often have a constant memory complexity. More in Appendix B.3.

With such efforts, our experiment report should be reasonably comparable among similar compression ratios. Though we emphasize that our additional alignment effort will not resolve the pretraining difference among different backbone models — where an aligned comparison here can only be done by training different models from scratch, which will induce drastic computation costs and can only provide coverage on fully transparent transformer-based LLMs like Pythia (Biderman et al., 2023), OpenLLaMA (Geng and Liu, 2023), or LLM360 (Liu et al., 2023), where weight-only open-source models like Llama (Touvron et al., 2023; AI@Meta, 2024) and Mistral (Jiang et al., 2023a) can not be included due to the lack of reproducible training procedure and resource.

3.3 Results and Discussion

We showcase our main results in a category-based fashion in Table 2 and **refer our readers to Appendix D for many more additional results**. Table 2 highlights the per-task-category performance of different long context-capable methods on Meta-Llama-3-8B-Instruct (AI@Meta, 2024) and Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a), as well as several other covered linear and mixed models. Based on all of our obtained results, we made the following observations.

OB ❶ Keeping the prefill process uncompressed is crucial for performance maintenance. This is because the KV cache for all prompt tokens is generated during the prefill stage. If we apply any compression at this stage, it will make the represen-

tation of said prompt in later layers inaccurate due to lossy `forward()` activation, leading to worse results when generating the output tokens. For instance, KIVI (Liu et al., 2024b), FlexGen (Sheng et al., 2023), and H_2O (Zhang et al., 2024d) do not employ any compression operation during the prefill stage, which often leads to much better results than methods which do compress within (or even before) the prefill stage, namely StreamingLLM (Xiao et al., 2023), InfLLM (Xiao et al., 2024), and LLMLingua2 (Pan et al., 2024).

That being said, we note this observation is likely limited to "long input" type of tasks, as all evaluated tasks in our work are considered "long input, short output" (like passkey retrieval from Mohtashami and Jaggi (2023)), but not "long generation" (like multi-round conversation (Li et al., 2023b; Wu et al., 2023), fiction writing (Yang et al., 2022), or long code generation (Roziere et al., 2023)), where compressing the input during the prefill stage will naturally carry more influence than compression during the decoding stage. More on this in Section 5.

OB ❷ Quantization methods can often achieve reliable performance across all task categories, yet token dropping approaches excel on some specific types of tasks (e.g., coding). We find that KV cache quantization techniques like FlexGen (Sheng et al., 2023) and KIVI (Liu et al., 2024b) tend to perform decently across all evaluated tasks. This is an intuitive finding, given quantization techniques do not evict any token completely, avoiding the possibility of dropping task-influential tokens by accident (e.g., one can imagine forging tokens around the needle insertion in the needle-in-the-haystack tasks (Mohtashami and Jaggi, 2023) will surely be damaging, especially if such eviction happens during the prefill stage). The trade-off of such globally acceptable performance of KV cache quantization methods is their memory footprints *must* grow with the sequence length, unlike token dropping approaches or linear-time sequence models, where a constant memory footprint is possible.

On the other hand, several featured token dropping methods showcased excellent performance on some specific subtasks. For example, StreamingLLM (Xiao et al., 2023) and H_2O (Zhang et al., 2024d) tend to perform exceptionally well on code-related tasks, with Figure 2 and Figure 26 demonstrating perfect performance retention across various compression ratios upon the majority of featured LLMs; whereas InfLLM (Xiao et al., 2024) — another token dropping methods that basically does KV cache retrieval of middle tokens on top of StreamingLLM — tend to deliver a more steady performance across all tasks without drastic shortcoming, with an extra advantage of being stronger under the needle test than StreamingLLM.

Conversely, hard prompt compression methods like LLMLingua2 (Pan et al., 2024) perform the worst on the needle test across all KV cache-required methods — which is, once again, a well-expected finding as if one deletes the needle information within the input, the LLM will certainly not be able to answer the retrieval-required question correctly. LLMLingua2 performs modestly behind all featured KV cache-required methods in terms of LongBench (Bai et al., 2023b) tasks, though with the advantage of being model agnostic and can be theoretically applicable to black-box models with limited access.

OB ❸ Mixing with attention can greatly improve the long context capability of linear-time sequence models. We observe that hybrid models like RecurrentGemma (Botev et al., 2024) can result in good performance improvement over pure linear-time sequence models like Mamba (Gu and Dao, 2023) or Mamba-Chat (Mattern and Hohr, 2023) in terms of all evaluated tasks (Table 2). This indicates the potential of hybrid architectures due to the promising performance gain with an often acceptable increase in memory footprint.

OB ❹ Needle-in-a-haystack test remains challenging for KV cache-free or prefill time compression methods. As demonstrated in Figure 3, which features the best methods from each school of approaches: KIVI by Liu et al. (2024b) (quantization), InfLLM by Xiao et al. (2024) (token dropping), LLMLingua2 by Pan et al. (2024) (prompt compression), Mamba-2.8B by Gu and Dao (2023) (linear-time sequence models), and RecurrentGemma-9B-it by Botev et al. (2024) (mixed architectures), we observe that compression during prefill or KV cache-free methods often struggle to maintain good retrieval performance as the baseline methods. While we believe different architectural or method designs do play a role here, we emphasize that unaligned pretraining recipes among different models, as well as the disparity of model sizes, are also certainly some strong influencing factors. For example, while not featured in

our work, LongMamba (Zhang, 2024) — a fine-tuned version of Mamba-2.8B (Gu and Dao, 2023) with long context focuses - tends to have much better needle performance.

Additionally, we note that we purposely decide to feature InfLLM (Xiao et al., 2024) instead of H_2O (Zhang et al., 2024d) in Figure 3 as a representation of the token dropping school, despite H_2O having an objectively much better needle result in Table 2 (100% vs 20.7% for 4× compression). This decision is made because our needle test requires the model to correctly answer a 7-digit passkey, where the ending of the instruction prompt is "`What is the pass key? The pass key is `" (Appendix A.2), leading the model-in-question likely to answer the first several digits of the passkey as the first generated token. This, combined with the fact that H_2O does not evict tokens during prefill time, often means an H_2O-powered model can get the first several digits (usually at least three, due to the design of tokenizers) of the passkey right for free, as no compression has happened for decoding the first token, and most transformer-based baseline models — like the Llama-3-8B-Instruct featured in Figure 3 — are able to get the full 7-digit passkey right under no compression. We confirmed H_2O's perfect needle performance on Llama-3-8B-Instruct showed in Table 2 and Figure 14 is indeed more of a product of this prompt template and the 7-digit passkey task configuration instead of its innate excellence in retrieval capability; as should we expand the passkey length to 64-digit while keeping everything else the same, H_2O's performance drops drastically (from 100% to 35.0% for 4× compression), where methods like KIVI (Liu et al., 2024b) and InfLLM (Xiao et al., 2024) tend not to experience such significant of a performance drop (100% to 91.0% for KIVI-2bit; 20.7% to 19.0% for InfLLM 4× compression), as shown in Figure 21, 22, and 23. **Due to page limitations, we analyze more observations in Appendix E.**

4 Challenges and Opportunities

In this section, we share our insights regarding different long context challenges and highlight several opportunities derived from our benchmarking observations.

How to effectively reduce prefill time and footprint? Based on our empirical observations, most KV cache compression methods struggle to make the prefill stage efficient without compromising performance (**OB ❶**), which calls for investments in more performant prefill-time compression methods. However, other than the performance requirement on accuracy-like metrics, prefill-time compression methods are entangled with non-trivial technical comparability challenges. Recall that FlashAttention (FA) (Dao et al., 2022) is inevitable during the prefill stage to improve hardware utility, with the key spirit of FA being to avoid the generation of a full attention matrix. Thus, methods that rely on the availability of a full attention matrix cannot be easily integrated. Therefore, we advocate future research on prefill-time compression methods with FA compatibility in mind.

How to build efficient yet long context-capable architectures? We empirically observe that pure linear-time sequence models that mix input tokens together struggle with information retrieval (**OB ❸**), where some sort of attention mechanism provides visible improvements (**OB ❷**). Therefore, an important future direction is to explore how to efficiently combine attention layers with linear-time sequence model layers and determine the optimal number of attention layers needed to achieve an ideal performance-efficiency balance.

How to cash-in real-world efficiency? Different methods often have varying levels of optimization while being comparable in theoretical efficiency, meaning whether a method is practically efficient in real-world application is highly related to factors like the *Ease of Optimization* (e.g., quantization is well-studied and easy to optimize, while some unstructured methods will involve extra challenges (Liu and Wang, 2023)) and *Compatibility with Established Software or Hardware Frameworks* (e.g., compatibility with FlashAttention, as mentioned above). Based on these factors, it is challenging to provide a fair apple-to-apple comparison regarding efficiency. Researchers should keep this challenge in mind and develop efficient yet long context-capable methods.

5 Conclusion

Our benchmark fills a critical gap by evaluating 10+ methods across 65 settings, uncovering new insights on long context-capable approaches. We also provide a minimalistic and extensible package for reproducible research.

Acknowledgments

This research was partially supported by NSF Awards ITE-2429680, IIS-2310260, OAC-2112606, and OAC-2117439. Furthermore, this work was supported by the US Department of Transportation (USDOT) Tier-1 University Transportation Center (UTC) Transportation Cybersecurity Center for Advanced Research and Education (CYBER-CARE) grant #69A3552348332.

This work also made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University (CWRU). We give special thanks to the CWRU HPC team for their prompt and professional help and maintenance. The views and conclusions in this paper are those of the authors and do not represent the views of any funding or supporting agencies.

Limitations and Potential Risks

Despite our best efforts to cover a wide range of long context-capable approaches across many backbone models, our benchmark work will inevitably lack the inclusion of some eligible and interesting methods, certain worthwhile tasks, or particular setups that are reflective of our benchmarking goal due to limited manpower and computing resources. Specifically, we recognize that we only benchmark on models with $<$10B parameters[2] and our tasks are more focused on long input but not long generation, with the latter also being an important, though less mature aspect of long context evaluation due to the open-ended nature of prolonged generation tasks.

In terms of potential risks, while we aim to provide a comprehensive view of feature methods and tasks, we caution our readers to directly adopt our empirical conclusion without proper evaluation under high-stake scenarios.

References

AI@Meta. 2024. Llama 3 model card.

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. *arXiv preprint arXiv:2305.13245*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023a. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023b. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Aleksandar Botev, Soham De, Samuel L Smith, Anushan Fernando, George-Cristian Muraru, Ruba Haroun, Leonard Berrada, Razvan Pascanu, Pier Giuseppe Sessa, Robert Dadashi, et al. 2024. Recurrentgemma: Moving past transformers for efficient open language models. *arXiv preprint arXiv:2404.07839*.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. 2024. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E Gonzalez, et al. 2024. Chatbot arena: An open platform for evaluating llms by human preference. *arXiv preprint arXiv:2403.04132*.

Yu-Neng Chuang, Tianwei Xing, Chia-Yuan Chang, Zirui Liu, Xun Chen, and Xia Hu. 2024. Learning to compress prompt in natural language formats. *arXiv preprint arXiv:2402.18700*.

[2]Though part of it is to align with linear-time sequence models, which are often $\leq$8B.

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

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning*.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. 2024. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*.

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

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. 2024. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. 2024. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*.

Yao Fu. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv preprint arXiv:2405.08944*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2023. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*.

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama.

Olga Golovneva, Tianlu Wang, Jason Weston, and Sainbayar Sukhbaatar. 2024. Contextual position encoding: Learning to count what's important. *arXiv preprint arXiv:2405.18719*.

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

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *arXiv*.

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. 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. 2023b. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same task, more tokens: the impact of input length on the reasoning performance of large language models. *arXiv preprint arXiv:2402.14848*.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023a. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023b. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Jingyao Li, Han Shi, Xin Jiang, Zhenguo Li, Hong Xu, and Jiaya Jia. 2024a. Quickllama: Query-aware inference acceleration for large language models. *Preprint*, arXiv:2406.07528.

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

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024c. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. 2024. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*.

Shiwei Liu and Zhangyang Wang. 2023. Ten lessons we have learned in the new" sparseland": A short handbook for sparse neural network researchers. *arXiv preprint arXiv:2302.02596*.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. 2023. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024a. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. 2022. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*.

Justus Mattern and Konstantin Hohr. 2023. Mamba-chat. GitHub.

Amirkeivan Mohtashami and Martin Jaggi. 2023. Landmark attention: Random-access infinite context length for transformers. *arXiv preprint arXiv:2305.16300*.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. *arXiv preprint arXiv:2304.08467*.

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

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. 2024. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Ruhle, 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*.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. 2023. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. 2024. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

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

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models (2023). *URL http://arxiv.org/abs/2307.08621 v1*.

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. *arXiv preprint arXiv:2405.05254*.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.

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

Guanchu Wang, Junhao Ran, Ruixiang Tang, Chia-Yuan Chang, Yu-Neng Chuang, Zirui Liu, Vladimir Braverman, Zhandong Liu, and Xia Hu. 2024a. Assessing and enhancing large language models in rare disease question-answering. *arXiv preprint arXiv:2408.08422*.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Yicheng Wang, Jiayi Yuan, Yu-Neng Chuang, Zhuoer Wang, Yingchi Liu, Mark Cusick, Param Kulkarni, Zhengping Ji, Yasser Ibrahim, and Xia Hu. 2024b. Dhp benchmark: Are llms good nlg evaluators? *arXiv preprint arXiv:2408.13704*.

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

Haoyi Wu and Kewei Tu. 2024. Layer-condensed kv cache for efficient inference of large language models. *arXiv preprint arXiv:2405.10637*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*.

Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. *arXiv preprint arXiv:2210.06774*.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2023. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*.

Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. 2022. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10819–10829.

Jiayi Yuan, Ruixiang Tang, Xiaoqian Jiang, and Xia Hu. 2023. Large language models for healthcare data augmentation: An example on patient-trial matching. In *AMIA Annual Symposium Proceedings*, volume 2023, page 1324. American Medical Informatics Association.

Amir Zandieh, Majid Daliri, and Insu Han. 2024. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *arXiv preprint arXiv:2406.03482*.

Lei Zhang, Yunshui Li, Ziqiang Liu, Jiaxi yang, Junhao Liu, Longze Chen, Run Luo, and Min Yang. 2024a. Marathon: A race through the realm of long context with large language models. *Preprint*, arXiv:2312.09542.

Peiyuan Zhang. 2024. Longmamba. `https://github.com/jzhang38/LongMamba`.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024b. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. 2024c. ∞bench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024d. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.

A Details about Datasets

A.1 Details Regarding LongBench

For the aforementioned task (1)-(6), we adopt the implementation and benchmark setting of LongBench (Bai et al., 2023b); here's a more detailed introduction of tasks.

The long context benchmarking tasks are categorized into several types: Multi-document QA, Single-document QA, Summarization, Few-shot learning, Synthetic tasks, and Code tasks. Each task has specific metrics for evaluation, such as the F1 score, ROUGE-L, and Accuracy. The average length of most tasks ranges from 5k to 15k, and each task has 200 datapoints, except for MultiFieldQA (150), LCC (500), and RepoBench-P (500).

Single-document QA tasks include MultiFieldQA, NarrativeQA, and Qasper, each requiring the comprehension and extraction of information from lengthy texts. Multi-document QA tasks like HotpotQA, 2WikiMQA, and Musique require answering questions based on multiple documents. Summarization tasks, such as GovReport, MultiNews, and QMSUM, involve condensing long documents into concise summaries evaluated using Rouge-L. Few-shot tasks, including TriviaQA, SAMSum, and TREC, provide limited examples to guide the model in answering questions or categorizing data. Synthetic tasks like PassageRetrieval and PassageCount simulate real-world scenarios where models must identify relevant paragraphs or count distinct passages within a repetitive text. Code tasks such as LCC and RepoBench-P assess the model's ability to predict subsequent lines of code in various programming languages, emphasizing the use of cross-file dependencies.

Overall, LongBench's diverse tasks are meticulously designed to push the boundaries of long-context processing, providing a robust benchmark for assessing advanced language models.

In our benchmark, we purposely omit the results of PassageCount, as LLMs often do not count correctly even in relatively short contexts (Golovneva et al., 2024). All models and methods exhibit poor performance (i.e., less than 10% accuracy), making the average performance unreliable with such an outlier included.

A.2 Details Regarding Needle-in-a-Haystack Test

Needle-in-a-haystack (NIAH) is a style of synthetically generated stress test aiming to evaluate the information retrieval capability of language models. NIAH tasks often introduce a piece of key information that is inserted into unrelated background texts of various lengths and at various positions. To the best of our knowledge, the first two widely adopted versions of this task are proposed by Mohtashami and Jaggi (2023) and Greg Kamradt. Specifically, Mohtashami and Jaggi (2023) inserts a piece of key information formatted like "`The pass key is <PASS KEY>. Remember it. <PASS KEY> is the pass key`" into the different lengths of unrelated background texts filled by repetition of "`The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.`" — this task is often known as the passkey retrieval task. Yet, Greg Kamradt's version of NIAH inserts a sentence like "`The best thing to do in San Francisco is eat a switch and sit in Dolores Park on a sunny day.`" Under both tasks, the LLM-in-question is then asked to answer a question that would require it to retrieve such a piece of inserted information successfully.

Given the vast variants of such NIAH tasks (gkamradt, Arize-ai, Levy et al. (2024); Mohtashami and Jaggi (2023); Reid et al. (2024); Hsieh et al. (2024)) existing in the community, we clarify the formation of our needle task as the following, which largely follows the passkey retrieval prompt template of Mohtashami and Jaggi (2023); Wang et al. (2024a) but using 7-digit passkey and Paul Graham Essays[3] as the background filler, as set forth in Arize-ai and Reid et al. (2024):

```
There is an important info hidden inside a lot of
irrelevant text. Find it and memorize them. I will
quiz you about the important information there.
<prefix filled by Paul Graham Essays>
The pass key is <7-DIGIT PASS KEY>. Remember it.
<7-DIGIT PASS KEY> is the pass key.
<suffix filler>
What is the pass key? The pass key is
```

B Detailed Experiment Setup

B.1 LongBench Setting

For baseline (no compression) performance, we follow the truncation settings in the LongBench

[3] https://paulgraham.com/articles.html

official implementation as below in Table 3.

Table 3: Maximal prompt length in LongBench of different benchmarks.

Model	max_length
Meta-Llama-3-8B-Instruct	7,500
Mistral-7B-Instruct-v0.2	31,500
LongChat-7b-v1.5-32k	31,500

We note that following the official implementation of LongBench (Bai et al., 2023b), for prompts that exceed the `max_length` specified in Table 3, they will be middle-truncated by preserving the first and last `max_length`/2 tokens.

For a fair comparison, the LongBench inputs of prefill-time compression methods like InfLLM (Xiao et al., 2024) and StreamingLLM (Xiao et al., 2023) are not truncated, but their maximum cache budget is capped at the respective base model `max_length` $\times$ compression ratio. Namely, suppose InfLLM is evaluated on LongBench tasks with a base model of Mistral-7B-Instruct-v0.2 and under a compression ratio of $2\times$, its maximum KV cache budget would be equivalent to $31,500/2 = 15,750$ tokens (or the full prompt length/compression ratio, if such given prompt has a lower length than 31,500 tokens). The difference lies in that methods like InfLLM can decide where to allocate such budget across the full, non-truncated prompt, whereas methods like H_2O (Zhang et al., 2024d) and KIVI (Liu et al., 2024b) are only given the middle-truncated prompt at the first place due to such method do not conduct compression during the prefill stage. We refer our readers to Appendix B.3 for detailed settings regarding each compression method.

B.2 Needle-in-a-Haystack Setting

Following the designs of Mohtashami and Jaggi (2023) and Hsieh et al. (2024), we adopt the passkey retrieval task formulated in Appendix A.2 as our needle test. For granularity, we evaluate the LLM-in-question against 10 different sequence lengths uniformly spanning from `512` to `20480` words and in 10 different depths from the start to the end of the input. For each length-depth combination, we iterate the test 3 times with 3 randomly generated `<7-DIGIT PASS KEY>`. We highlight the length of our needle test — `20480` — is in terms of the number of words, but not the number of tokens, as different models might employ tokenizers with different efficiency, where an aligned input construction should be maintained for proper cross model comparison (which is inevitable given the involvement of linear-time sequence models and their hybrids). `20480` words in our needle test usually converts to roughly 30.6k tokens with the tokenizer utilized in models like LongChat-7b-v1.5-32k (Li et al., 2023a), but only 27.2k tokens in models like Meta-Llama-3-8B-Instruct (AI@Meta, 2024) with a more efficient tokenizer.

We evaluated our needle test against three popular transformer-based language models (Mistral-7b-Instruct-v0.2 (Jiang et al., 2023a), LongChat-7B-v1.5-32K (Li et al., 2023a), Llama-8B-Instruct (AI@Meta, 2024)) as well as several other liner-time sequence models and hybrid architectures mentioned in Section 3.1. Given that Mistral-7b-Instruct-v0.2 and LongChat-7B-v1.5-32K come with a context window of 32k tokens, we feed our needle inputs into such models in a vanilla fashion, whereas for Llama-8B-Instruct, we enlarge its RoPE base theta (θ) (Su et al., 2024) setting to $32\times$ of its original size due to its limited 8k off-the-shelf context window.

B.3 Method-specific Setting

Linear-time sequence models and mixed architecture In our paper, we benchmark five pure or hybrid linear-time sequence models. While such models can theoretically achieve infinite context lengths, model performance is still expected to degrade when the context length exceeds the effective context length, which is typically the length used during the pretraining phase. The context lengths used in benchmarking LongBench (Bai et al., 2023b) are provided in Table 4. For our Needle-in-a-Haystack task (Mohtashami and Jaggi, 2023; Hsieh et al., 2024) defined in Appendix A.2, we uniformly set the maximum context length to `20480` words to ensure consistency and fair comparison across tasks.

Table 4: Effective context length and model size of the five linear time sequence models benchmarked in our paper.

Model	Eff. context length
Mamba-2.8B	2k
Mamba2-2.7B	2k
Mamba-Chat-2.8B	2k
RWKV-5-World-7B	4k
RecurrentGemma-2B-it	8k
RecurrentGemma-9B-it	8k

Quantization We benchmark two popular KV cache quantization methods: one 2bit quantization (KIVI-2) and two 4bit quantizations (KIVI-4 and FlexGen). For KIVI (Liu et al., 2024b), we use the official implementation[4], and for FlexGen (Sheng et al., 2023), we follow the group-wise quantization in the official codebase[5]. The group size for both KIVI and FlexGen is set to 32. We further set the residual length, which is unique to KIVI, as 128.

Token Dropping We evaluate three popular token dropping methods used for handling long contexts: StreamingLLM (Xiao et al., 2023), InfLLM (Xiao et al., 2024), and H_2O (Zhang et al., 2024d). In H_2O, there are two parameters for controlling the token dropping ratio: the heavy ratio and the recent ratio. The recent ratio controls the number of tokens preserved within the local window, while the heavy ratio controls the number of heavy-hitter tokens outside the local window. We set both the heavy ratio and recent ratio to the same values of 25%, 12.5%, 8.3%, and 6.25% of the total token length to achieve compression gains of 2×, 4×, 6×, and 8×, respectively, following the setting[6] set forth in H_2O's official implementation[7].

We emphasize that under this linear compression scheme utilized in H_2O, the KV cache size scales linearly with the input prompt length. On the other hand, StreamingLLM maintains a constant window size of "attention sinks" (i.e., front-most tokens) and recent tokens, making the size of the KV cache constant at all times (irrelevant to input length) in its original design. Thus, to hit a consistent compression rate that is reasonably comparable to other methods, we modify the total number of tokens retained in the StreamingLLM pipeline as the product of the target compression rate and the input length — i.e., for a prompt of 1,000 tokens, a StreamingLLM-empowered LLM with 2× compression rate would have a 500 tokens KV cache budget to distribute among its attention sink and most recent tokens. We ensure the ratio of attention sinks to recent tokens within the KV cache matches the ratio of 2% and 98%, according to its official configurations[8].

In addition to the attention sink and recent tokens, InfLLM (Xiao et al., 2024) incorporates the most relevant tokens from the middle of the context into the kept KV cache. We, therefore, preserve the ratio of attention sinks, middle tokens, and recent tokens as 2%, 32%, and 66%, respectively, again being faithful to its official configurations[9].

We borrowed our implementations of StreamingLLM and InfLLM from InfLLM's (Xiao et al., 2024) official repository[10] as this is the official implementation of InfLLM, yet it is endorsed by the lead author of StreamingLLM due to overlapped authorships.

Prompt Compression We evaluate LLMLingua[11] (Pan et al., 2024) on four different compression rates. K× for $K \in \{2, 4, 6, 8\}$ denotes that the compressor is restricted to compress the length into 1/K of the original length of long inputs.

C Related Works

The evaluation of LLM has been well studied (Chiang et al., 2024; Wang et al., 2024b). Given the importance of long context-capable LLMs, many related works try to quantify such capabilities, usually via means of purposing new, long context-focused datasets. For example, LongBench (Bai et al., 2023b) — which is also utilized in our work — provides a bilingual, multitask benchmark for long context understanding. Datasets like InfiniBench (Zhang et al., 2024c), LongICLBench (Li et al., 2024b), Marathon (Zhang et al., 2024a), and Ruler (Hsieh et al., 2024) all contribute their perspective in terms of long context evaluation via different collections of real or synthetic tasks.

Our work differs from the abovementioned prior arts as such arts mainly focus on producing long context evaluation datasets, where the included benchmarks — if any — are mostly evaluated on vanilla baseline models without any compression methods applied; where our work presents comprehensive results primarily highlighting **the comparison among different efficient but long context-capable approaches**. We cover 10+ long context-capable approaches under 60+ different settings. To the best of our knowledge, no prior art has benchmarked similar coverage of compression methods under a long context scenario as we do.

[4] https://github.com/jy-yuan/KIVI

[5] https://github.com/FMInference/FlexGen

[6] https://github.com/FMInference/H2O/blob/main/h2o_hf/README.md

[7] https://github.com/FMInference/H2O

[8] e.g., https://github.com/thunlp/InfLLM/blob/main/config/mistral-stream-llm.yaml

[9] e.g., https://github.com/thunlp/InfLLM/blob/main/config/llama-3-inf-llm.yaml

[10] https://github.com/thunlp/InfLLM

[11] https://github.com/microsoft/LLMLingua

D Extended Experimental Results

In this section, we present additional experimental results for LongBench and the needle tasks.

Table 5 shows all the LongChat-7B results on LongBench and the needle experiment. We present FlexGen (Sheng et al., 2023) results on three different LLMs in Figure 7. Additional H_2O (Zhang et al., 2024d) results for different compression ratios on Llama-3-8B, LongChat-7B-v1.5, and Mistral-7B-v0.2 can be found in Figure 14, 15 and 16 respectively.

We provide more visualization results on the needle task. For baseline performance for the three models in Figure 5. For InfLLM results on the LongChat and Mistral models, the results are listed in Figure **??** and 10. Figure 24 and 25 show the performance of quantization, token dropping, and prompt compression on Mistral and LongChat, respectively. Figure 26, 27 and 28 illustrates the effectiveness of different compression ratios across various subtasks in LongBench.

Finally, Table 6, 7, 8 and 9 show the detailed results for each task in LongBench.

We additionally have Figure 21, 22, and 23 to showcase the performance drop of H_2O (Zhang et al., 2024d) under the needle test with a 64-digit passkey as mentioned in OB ❹, in comparison to other methods.

E Extended Results and Discussion

A note on the "overtraining" recipe. In section 3.2, we briefly mentioned an "overtrained recipe." This is mostly referring to Llama-3-8B, which is trained on 15T tokens and is way beyond the optimal point according to Chinchilla scaling law (Hoffmann et al., 2022). This overtraining recipe is considered a main contributor to Llama-3's performance improvement.

We highlight it because most RNN/hybrid architectures are trying to outperform some fully transparent LLMs (ones we can reproduce the pretraining, in contrast to just having access to the trained weights) at a certain parameter scale with an identical training recipe — e.g., Mamba (Gu and Dao, 2023) to Pythia (Biderman et al., 2023) — where such LLMs-in-comparison do not employ this overtraining ingredient. This presents a gap in directly comparing the performance of weight-only open-sourced LLMs with fully transparent RNN/hybrids, and we alert our readers to be vigilant in drawing direct numerical comparisons.

A note on LLMLingua2 and coding tasks. LLMLingua2 (Pan et al., 2024) performs significantly worse in LCC compared to RopeBench-P, despite both being coding tasks (Table 6, Table 7, and Table 8). We hypothesize this is because LCC is a single file code completion task, where RopeBench-P prefixes the code modules according to the important statements of a certain file at a repository level. This potentially gives RopeBench-P a natural "outline," which can be favorable cues for hard prompt compression approaches like LLMLingua2 as these cues may drive the compression of different parts accordingly.

A note on Mamba-Chat. While Mamba-Chat (Mattern and Hohr, 2023) is presented as an instruction-tuned version of Mamba (Gu and Dao, 2023), it does not deliver much better performance than the original Mamba. Though much of this can be attributed to the particular instruction tuning recipe of Mamba-Chat, it suggests that supervised finetuning SSM models might require some extra considerations and careful monitoring.

A note on InfLLM with models utilizing condensing rotary embeddings. We noticed a significant performance improvement in InfLLM (Xiao et al., 2024) on models utilizing condensing rotary embeddings (e.g., InfLLM on LongChat (Li et al., 2023a) in Table 5 between the first and current version of our work). Upon investigation, we realize that the original InfLLM implementation does not take into account of the condensing rotary embedding technique[12] (namely, `position_ids`/ratio) — a simple RoPE-variant (Su et al., 2024) with long context handling in mind, often utilized in LongChat and Vicuna (Chiang et al., 2023) family of models — as InfLLM authors then shifted their focus out of the Vicuna family in their later versions. Upon updated implementation, we observe a decent performance boost on InfLLM with LongChat (Table 5).

A note on measured peak memory usage reports. During the rebuttal of this work, we promised our reviewers that we'd include real, code-measured, peak memory usage reports in the camera-ready. We then realized that HuggingFace Transformers involves some significant KV cache implementation changes around v4.42[13], resulting in up to 2x

[12] https://lmsys.org/blog/2023-06-29-LongChat/

[13] https://github.com/huggingface/transformers/pull/30536

saving in memory consumption just by changing its versions. Transformers v4.42 is not available by the time of our submission and is in conflict with some of the environment requirements of our featured methods. For this reason, we will postpone sharing this report. We aim to provide an update in our repository once we are able to bring some of our featured methods to Transformers v4.45+ (where another major memory-related update[14] has been done.

A note on having an alternative visualization than radar chart. Our work mainly employs radar charts to demonstrate the LongBench-related results. This decision is made based on the original choice of visualization utilized in the LongBench paper (Bai et al., 2023b), and the fact that radar chart is one of the most space-efficient visualization options — a welcoming character when we are trying to feature multiple methods under different compression ratios against various datasets. That being said, we recognize that radar charts can sometimes be misleading due to amplifying the delta between different readings, as each apex of the radar chart is defined by the highest number in that regard.

Typically, a bar chart is the next best option and is free from the abovementioned concerns. However, a full bar chart plot for all the experiments we conducted would be too massive, as we are looking at roughly 20 method-compression ratio settings per LLM, where each method is tested against 7 categories of datasets. Here, we provide Figure 4, a bar chart plot of Table 2. Specifically, the LongBench result was compared across different methods on Llama-3 and other model architectures.

[14] https://github.com/huggingface/transformers/pull/31292

Table 5: Performance of KV cache quantization, token dropping, and prompt compression methods on LongChat-7B in our benchmark.

Model	Method	Comp. Ratio	Single. QA	Multi. QA	Summ.	Few-shot	Synthetic	Code	**LB Avg.**	**Needle**
longchat-7b-v1.5-32k	Baseline	1.00×	31.1	23.9	26.7	63.8	30.5	54.9	38.5	96.3
	KIVI-2bit	5.05×	30.3	23.1	26.5	63.6	32.2	53.9	38.0	85.3
	KIVI-4bit	3.11×	31.1	24.2	26.8	63.9	31.5	54.3	38.5	96.3
	FlexGen-4bit	3.20×	31.3	23.8	27.0	62.8	31.5	53.4	38.2	94.7
	InfLLM-2x	2.00×	29.2	23.3	25.8	51.8	19.5	51.9	34.2	58.7
	InfLLM-4x	4.00×	24.2	24.0	24.4	49.7	9.5	46.8	31.4	34.0
	InfLLM-6x	6.00×	21.1	23.5	23.5	48.9	10.0	46.5	30.2	35.0
	InfLLM-8x	8.00×	20.1	21.7	22.6	45.3	6.0	46.2	28.5	25.7
	StreamLLM-2x	2.00×	23.9	21.8	24.2	51.2	25.0	49.3	32.5	47.7
	StreamLLM-4x	4.00×	19.9	22.1	22.3	49.0	13.5	52.0	30.5	32.3
	StreamLLM-6x	6.00×	19.6	21.4	20.7	47.5	10.5	51.4	29.4	21.3
	StreamLLM-8x	8.00×	17.5	21.4	19.5	44.4	10.0	51.9	28.2	19.7
	H_2O-2x	2.00×	27.6	22.1	24.6	62.6	30.5	57.8	37.1	56.7
	H_2O-4x	4.00×	26.2	21.9	21.9	61.9	28.5	55.2	35.7	28.7
	H_2O-6x	6.00×	25.7	21.3	21.0	62.1	28.0	53.3	35.0	19.7
	H_2O-8x	8.00×	25.1	21.0	19.8	61.6	28.5	51.5	34.3	14.3
	LLMLingua2-2x	2.00×	25.7	22.3	25.4	35.4	19.5	32.6	27.4	28.7
	LLMLingua2-4x	4.00×	23.8	20.6	23.5	31.6	5.5	31.9	24.5	3.3
	LLMLingua2-6x	6.00×	22.6	20.2	22.6	32.3	5.0	31.9	24.1	0.6
	LLMLingua2-8x	8.00×	21.3	19.5	21.9	32.9	6.5	32.5	23.9	0.0

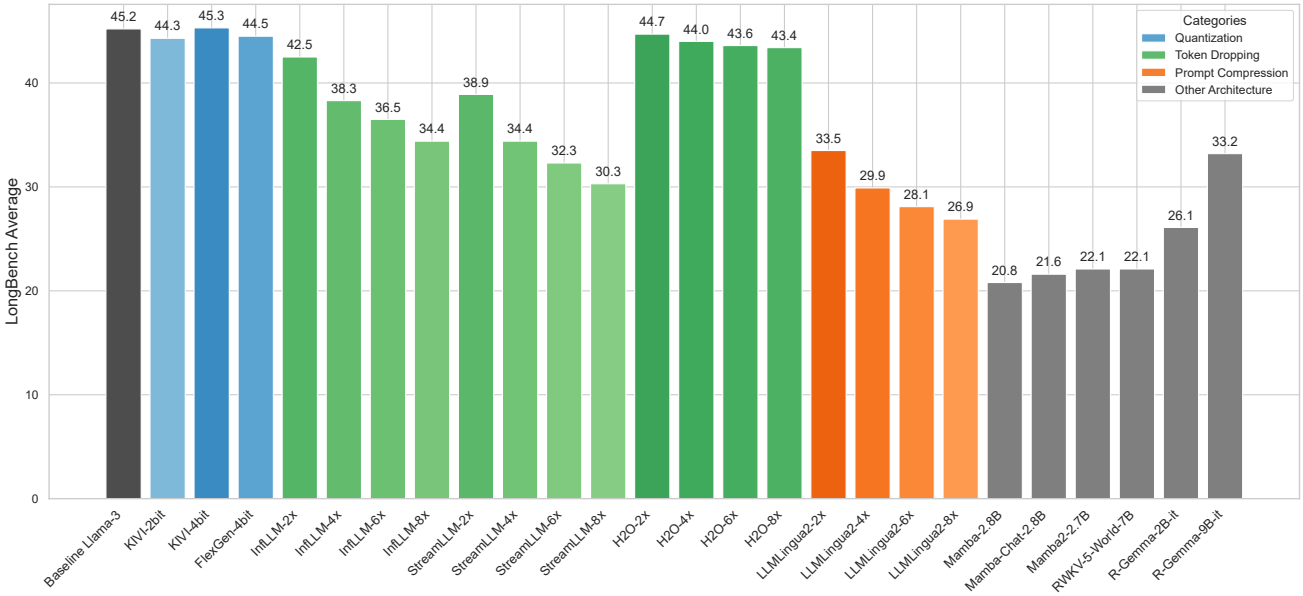
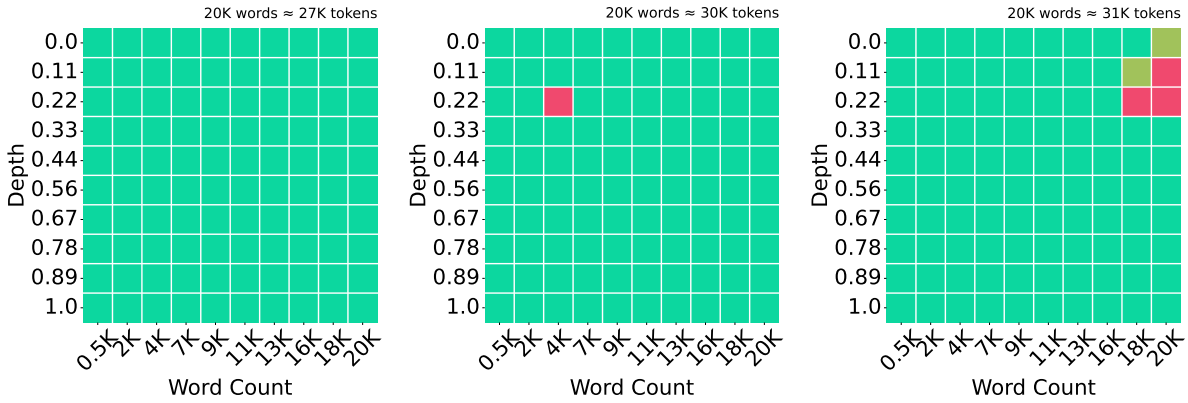
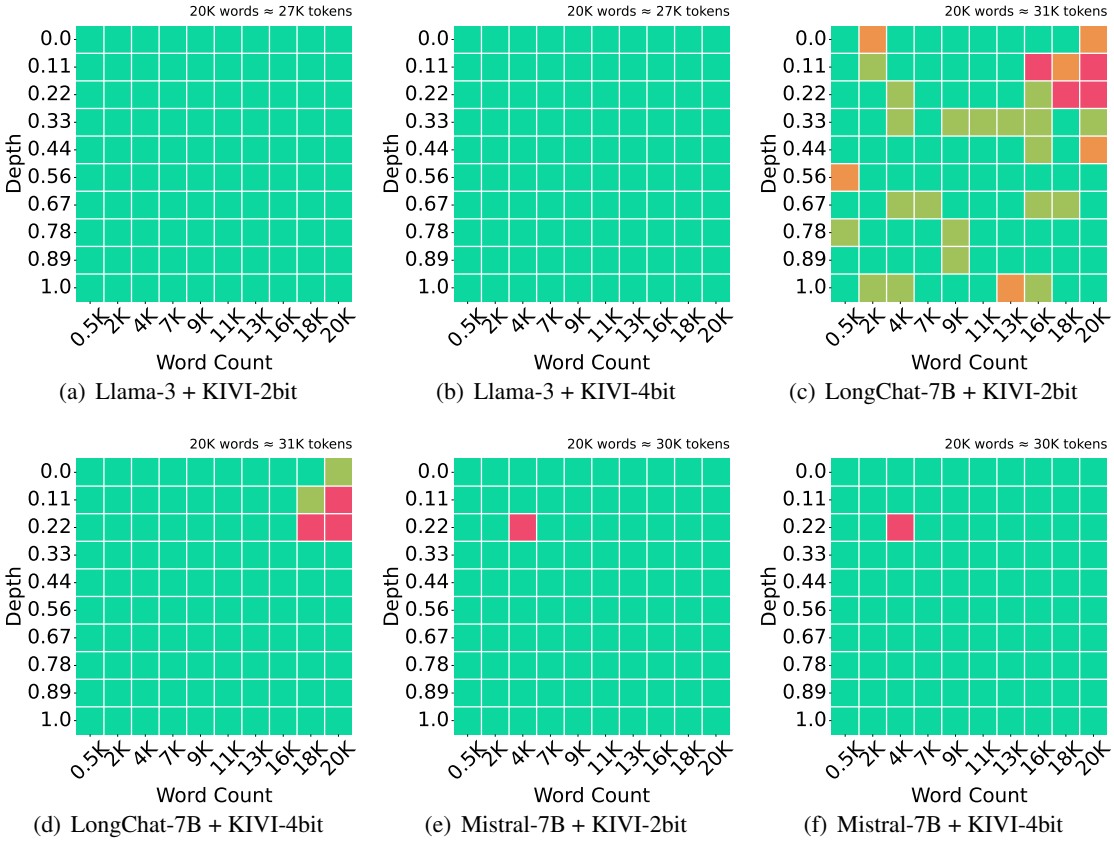


Figure 4: Performance of KV cache quantization, token dropping, prompt compression, and other architectures on LongBench.



(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32k

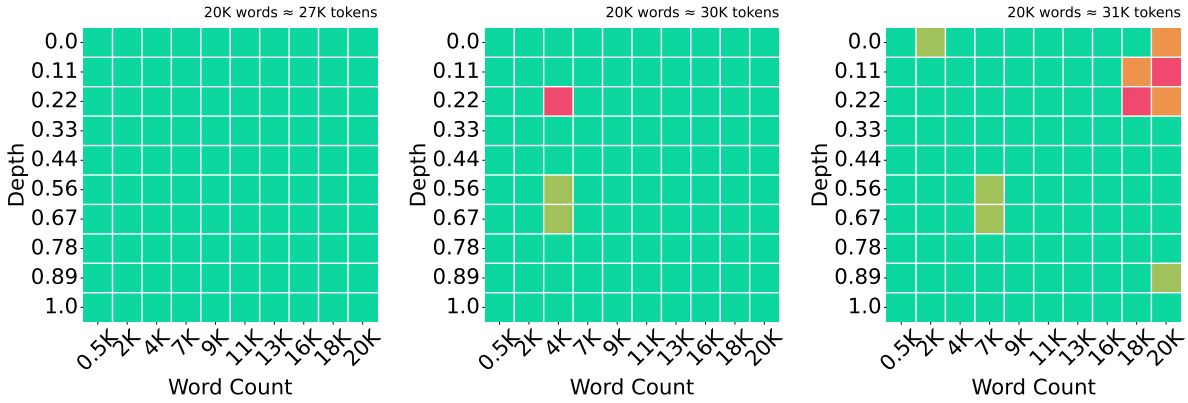
Figure 5: Baseline performance under needle test on three commonly used LLMs



(a) Llama-3 + KIVI-2bit (b) Llama-3 + KIVI-4bit (c) LongChat-7B + KIVI-2bit

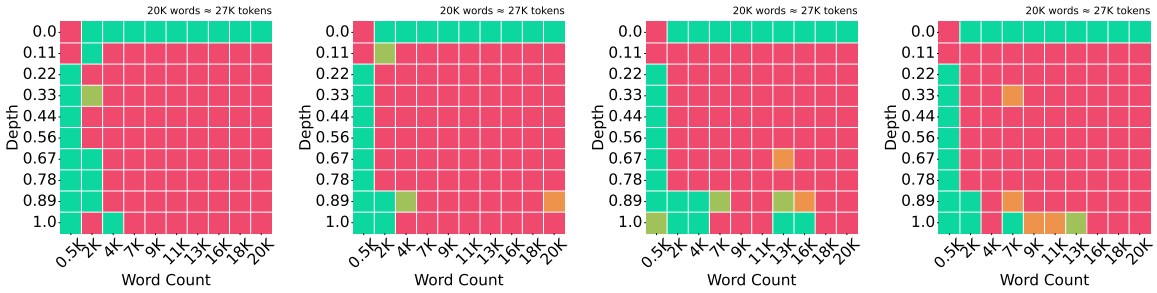
(d) LongChat-7B + KIVI-4bit (e) Mistral-7B + KIVI-2bit (f) Mistral-7B + KIVI-4bit

Figure 6: KIVI performance under needle test on three commonly used LLMs with 2-bit and 4-bit quantization



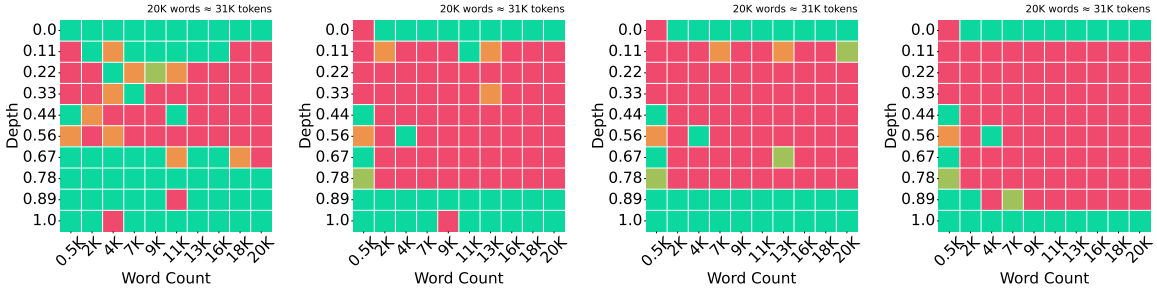
(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32k

Figure 7: FlexGen performance under needle test on three commonly used LLMs



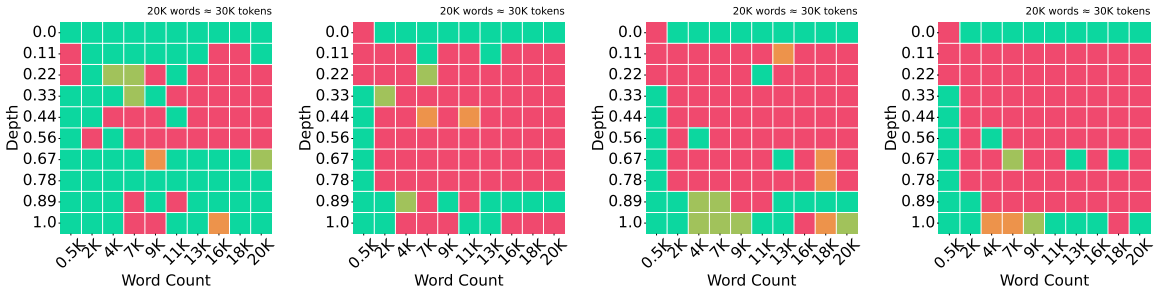
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 8: InfLLM on Llama-3-8B-Instruct with 4 different compression rates under needle test



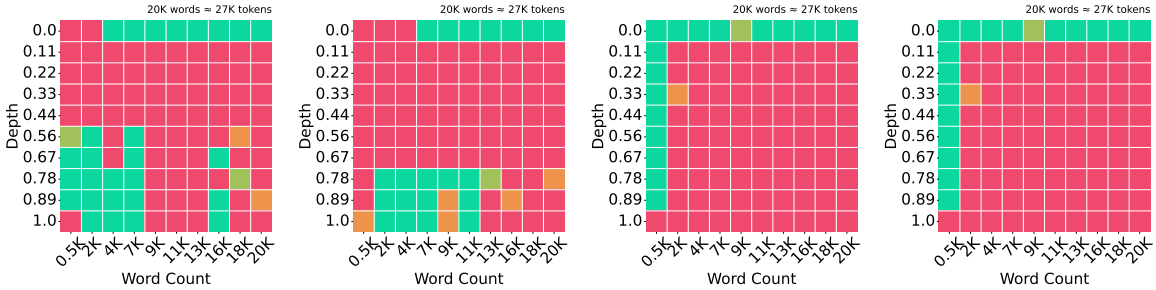
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 9: InfLLM on LongChat-7B-v1.5-32k with 4 different compression rates under needle test



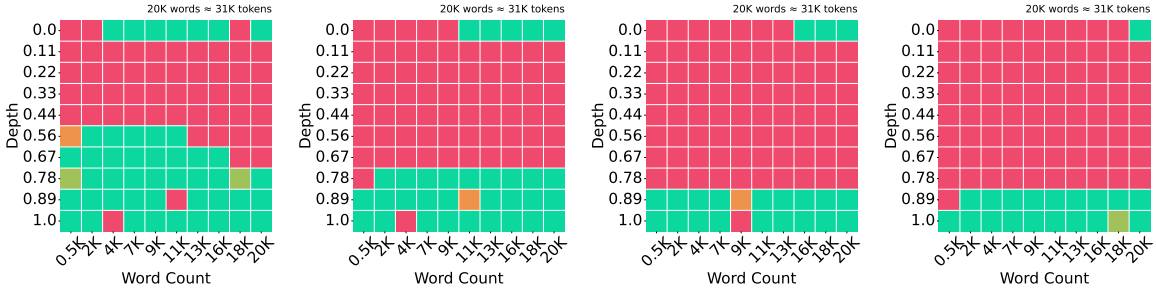
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 10: InfLLM on Mistral-7B-v0.2-Instruct with 4 different compression rates under needle test



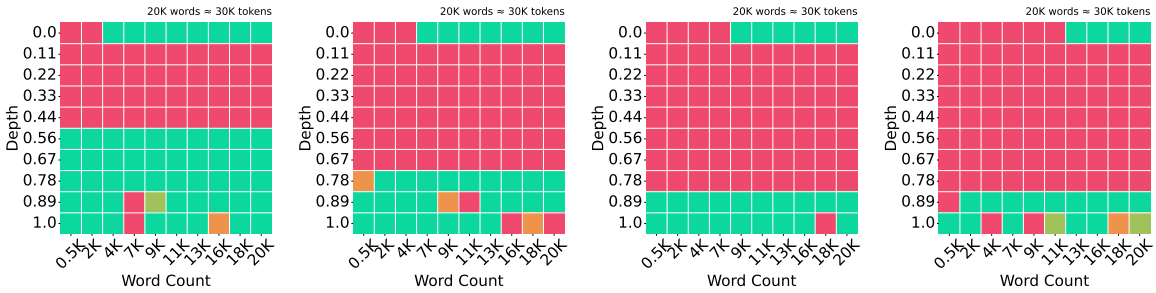
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 11: StreamingLLM on Llama-3-8B-Instruct with 4 different compression rates under needle test



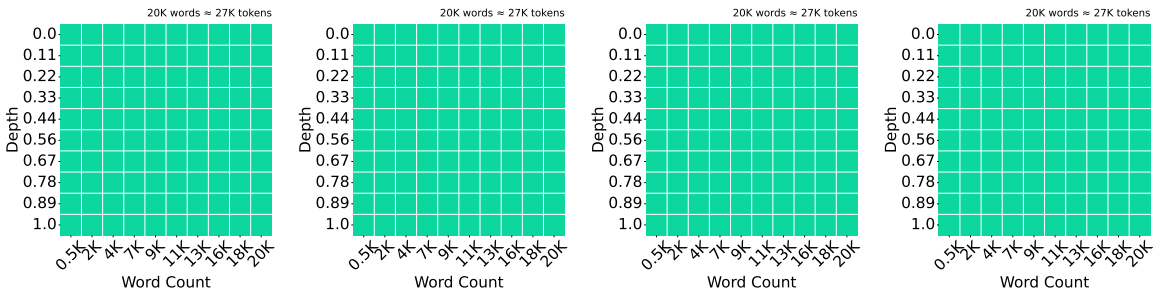
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 12: StreamingLLM on LongChat-7B-v1.5-32k with 4 different compression rates under needle test



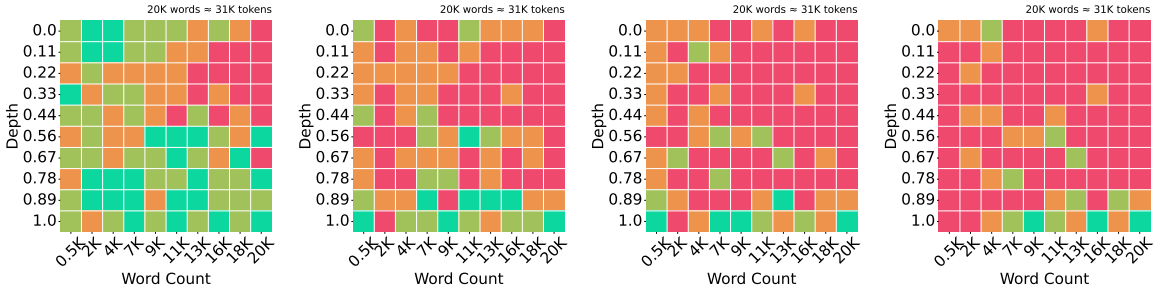
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 13: StreamingLLM on Mistral-7B-v0.2-Instruct with 4 different compression rates under needle test



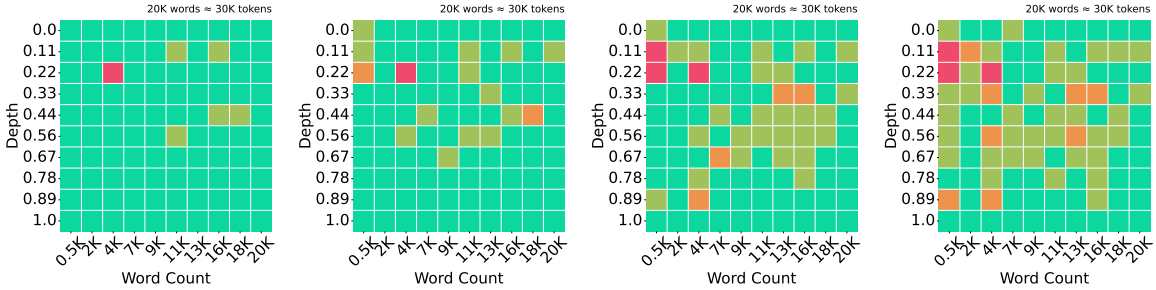
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 14: H_2O on Llama-3-8B-Instruct with 4 different compression rates under needle test



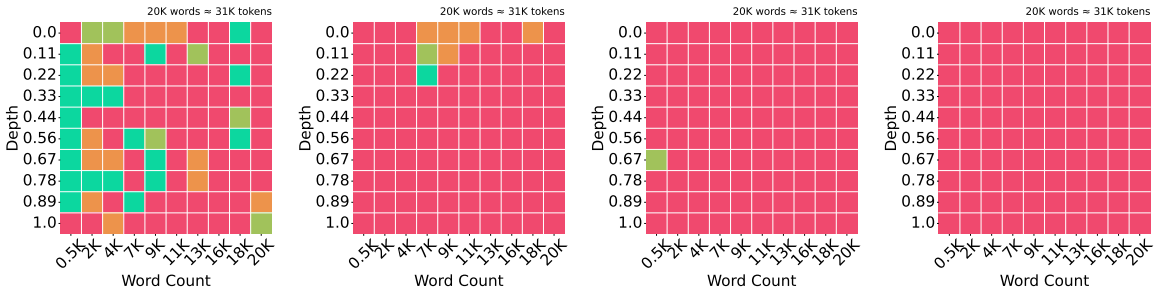
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 15: H_2O on LongChat-7B-v1.5-32k with 4 different compression rates under needle test



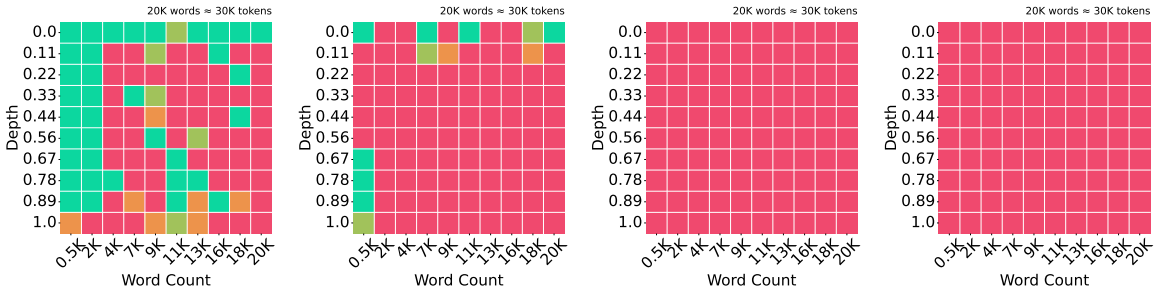
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 16: H_2O on Mistral-7B-v0.2-Instruct with 4 different compression rates under needle test



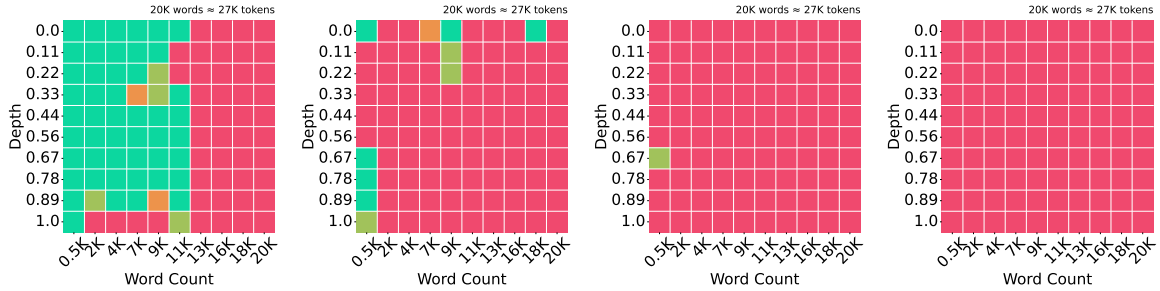
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 17: LLMLingua on LongChat-7B-v1.5-32k with 4 different compression rates under needle test



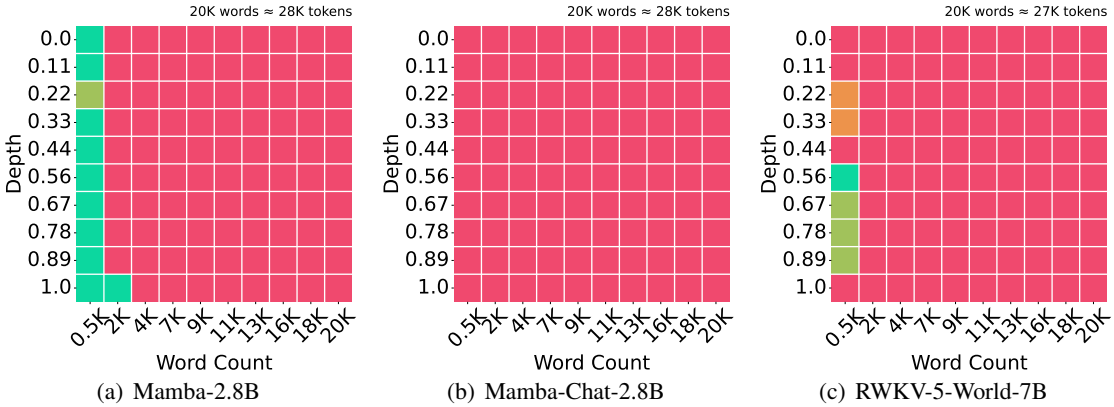
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 18: LLMLingua on Mistral-7B-v0.2-Instruct with 4 different compression rates under needle test

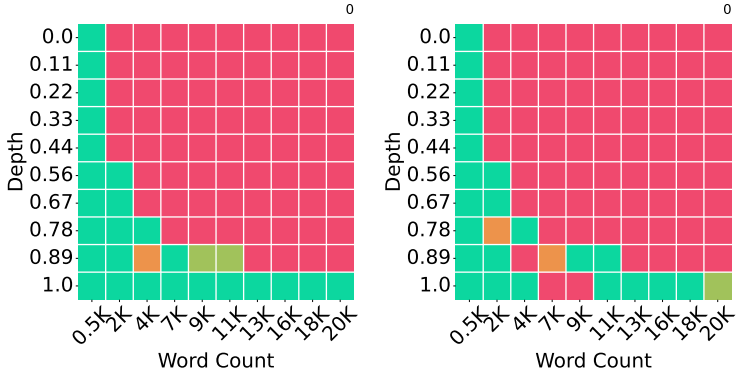


(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 19: LLMLingua on Llama-3-8B-Instruct with 4 different compression rates under needle test

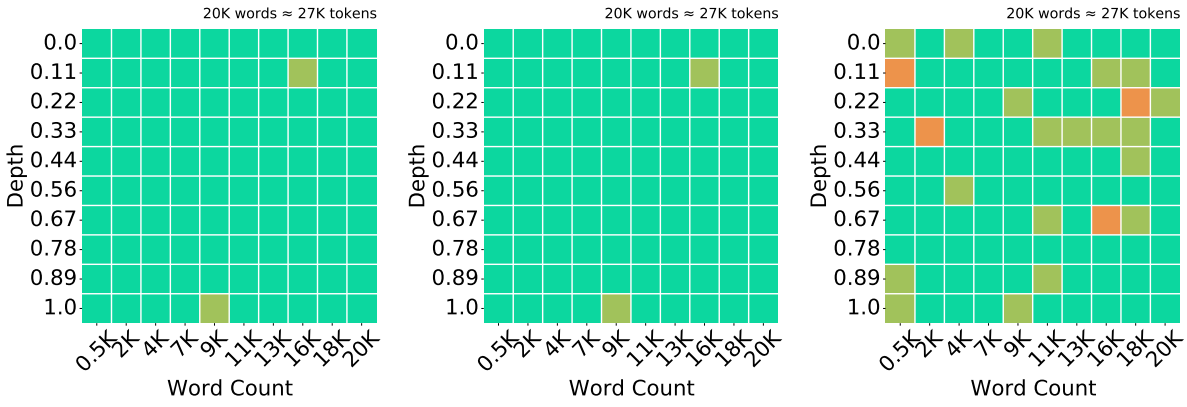


(a) Mamba-2.8B (b) Mamba-Chat-2.8B (c) RWKV-5-World-7B



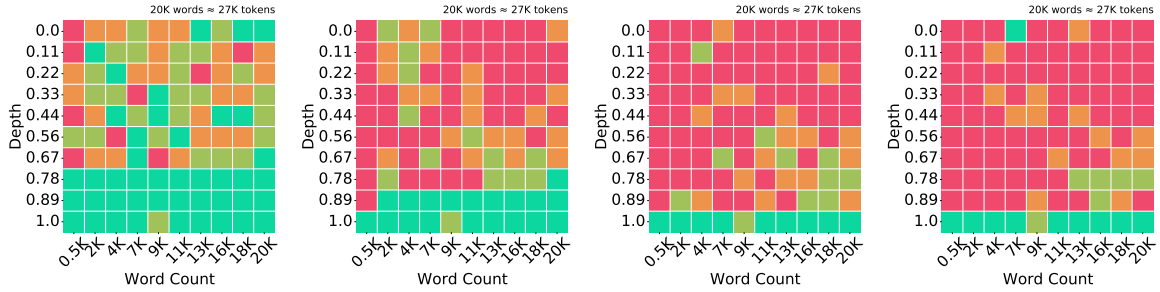
(d) RecurrentGemma-9B-it (e) RecurrentGemma-2B-it

Figure 20: Linear-time sequence models under needle test



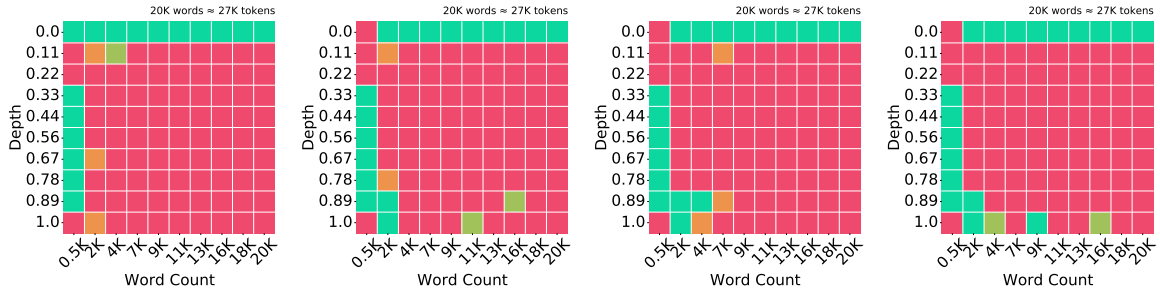
(a) Llama-3-8B-Instruct (99.3%) (b) Llama-3 + KIVI-4bit (99.3%) (c) Llama-3 + KIVI-2bit (91.0%)

Figure 21: Llama-3-8B-Instruct with no compression, as well as with 4bit and 2bit KIVI under needle test with 64-digit passkey. Overall accuracies are noted within parentheses.



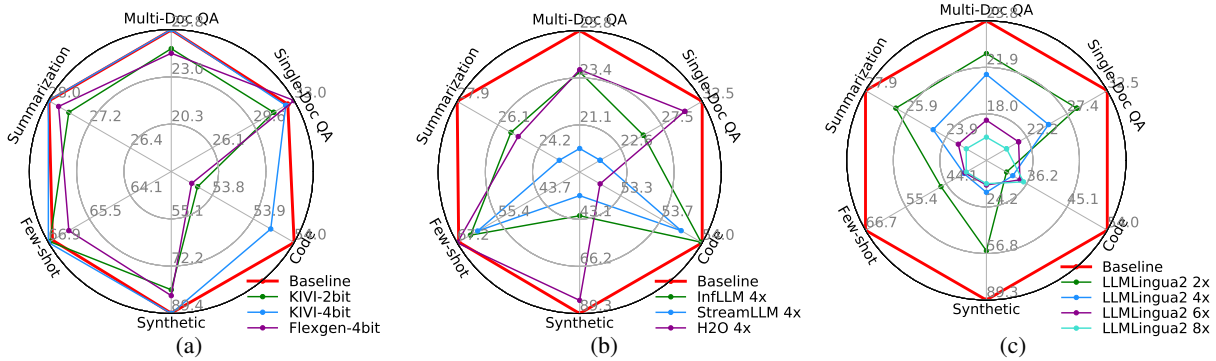
(a) 2x Compression (67.3%) (b) 4x Compression (35.0%) (c) 6x Compression (22.0%) (d) 8x Compression (18.7%)

Figure 22: H_2O on Llama-3-8B-Instruct with 4 different compression rates under needle test with 64-digit passkey. Overall accuracies are noted within parentheses.



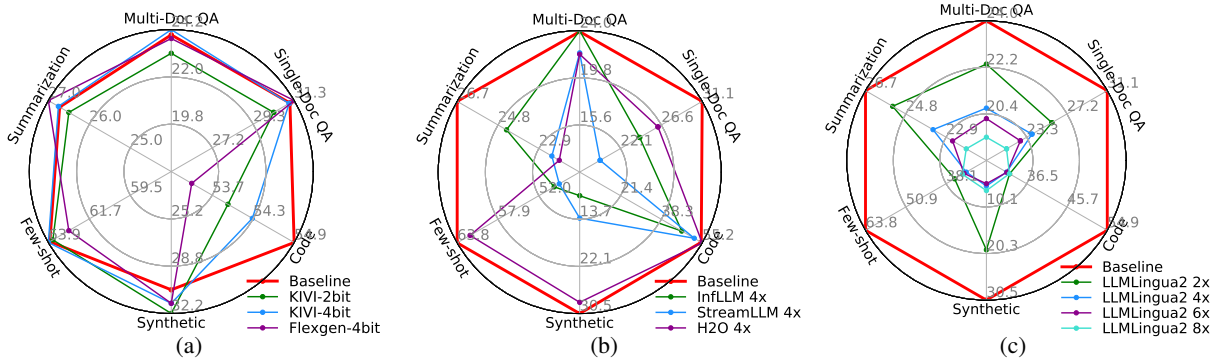
(a) 2x Compression (17.7%) (b) 4x Compression (19.0%) (c) 6x Compression (19.0%) (d) 8x Compression (19.3%)

Figure 23: InfLLM on Llama-3-8B-Instruct with 4 different compression rates under needle test with 64-digit passkey. Overall accuracies are noted within parentheses.



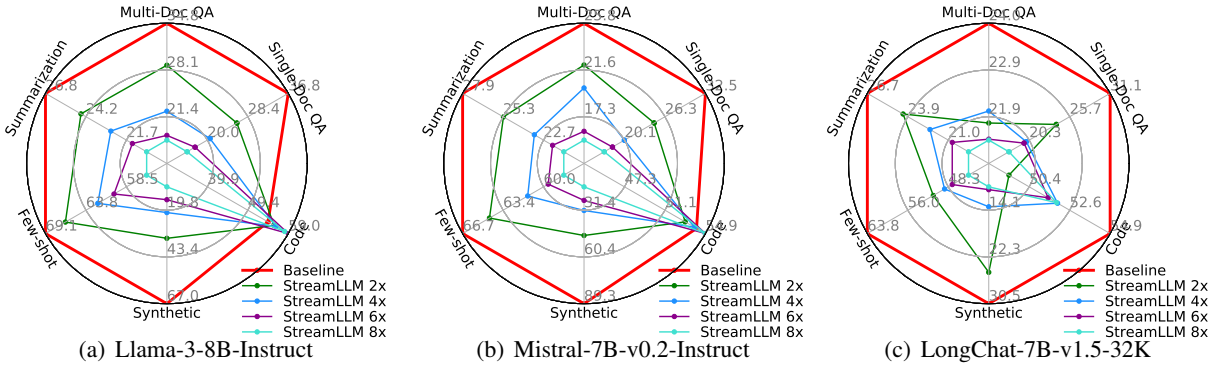
(a) (b) (c)

Figure 24: Mistral-7B-v0.2-Instruct with different compression methods (a) with Quantization; (b) with Token Dropping (c) with prompt compression.



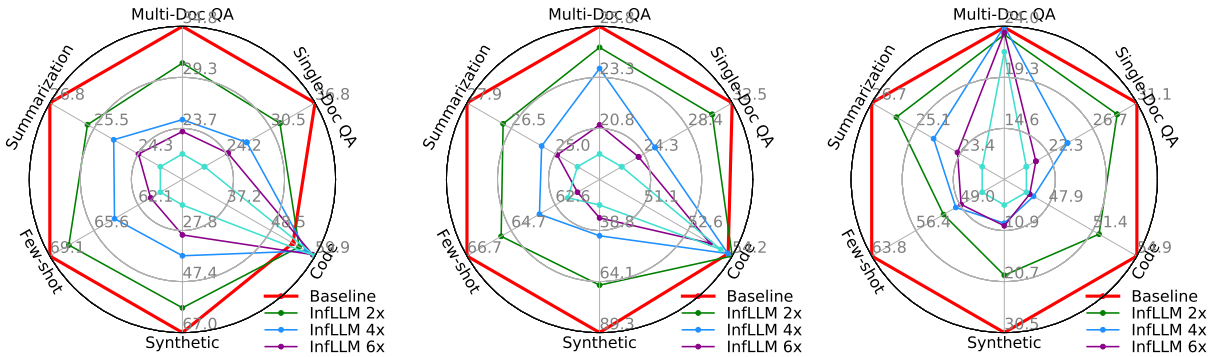
(a) (b) (c)

Figure 25: LongChat-7B-v1.5-32K with different compression methods (a) with Quantization; (b) with Token Dropping (c) with prompt compression.



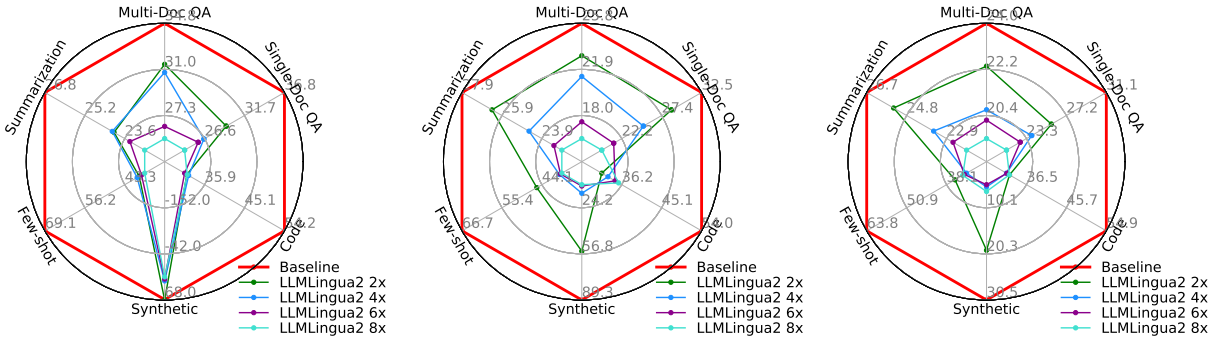
(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

Figure 26: StreamingLLM with different compression ratios on three commonly used LLMs.



(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

Figure 27: InfLLM with different compression ratios on three commonly used LLMs.



(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

Figure 28: LLMLingua with different compression ratios on three commonly used LLMs.

Table 6: Full report of different compression methods on meta-llama/Meta-Llama-3-8B-Instruct across 15 datasets in LongBench

LLM	Method \ Dataset	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code	Avg.
		NarrativeQA	Qasper	MultiFieldQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PassageRetrieval	LCC	RepoBench-P	
Meta-Llama-3-8B-Instruct	Baseline	21.7	44.3	44.5	46.6	36.4	21.5	29.9	22.6	27.7	74.0	90.6	42.7	67.0	57.2	51.2	45.2
	KIVI-2bit	21.4	43.1	44.2	46.8	37.0	20.6	29.8	22.1	27.5	74.5	90.5	42.5	67.5	50.8	46.7	44.3
	KIVI-4bit	21.0	44.8	44.6	47.0	36.5	21.4	30.1	22.5	28.0	74.5	90.3	43.1	66.5	57.3	52.0	45.3
	FlexGen-4bit	20.9	44.0	44.5	43.3	33.5	20.5	29.7	22.0	27.7	73.0	90.5	42.2	65.5	59.7	50.7	44.5
	InfLLM-2x	19.7	38.5	37.3	40.9	30.8	20.9	29.9	20.7	26.5	69.0	91.2	42.5	57.5	57.6	54.0	42.5
	InfLLM-4x	18.1	28.0	35.1	36.6	23.0	14.4	29.6	19.8	25.5	61.0	88.8	42.0	37.5	56.9	58.3	38.3
	InfLLM-6x	19.3	24.1	29.8	35.9	19.3	15.0	28.8	19.4	24.6	56.0	85.4	41.8	29.5	60.2	58.3	36.5
	InfLLM-8x	14.2	22.0	27.0	33.1	20.5	9.3	27.5	19.1	24.4	58.0	82.0	40.9	18.0	61.4	58.4	34.4
	StreamLLM-2x	17.3	33.5	27.6	37.0	30.3	19.0	28.1	20.1	25.5	68.0	90.4	41.1	34.0	55.0	56.2	38.9
	StreamLLM-4x	17.4	23.0	21.1	29.8	24.7	12.0	25.9	19.5	22.6	60.5	85.7	40.5	21.0	55.0	57.2	34.4
	StreamLLM-6x	15.7	18.7	17.8	26.1	19.3	10.7	24.7	18.6	20.7	58.0	82.2	40.2	14.5	59.4	58.5	32.3
	StreamLLM-8x	13.1	16.6	17.4	25.7	18.5	9.8	23.4	18.2	19.9	55.5	72.3	39.9	8.0	60.4	55.8	30.3
	H_2O-2x	21.5	42.6	43.2	46.4	36.5	21.5	28.1	22.1	26.0	74.0	90.6	42.8	66.0	57.2	51.6	44.7
	H_2O-4x	21.8	41.2	41.8	46.8	36.9	21.5	25.7	21.4	23.7	74.0	90.6	42.4	66.0	55.1	51.2	44.0
	H_2O-6x	21.5	38.3	41.8	46.8	36.8	21.7	24.5	21.1	22.5	74.0	90.5	42.6	66.0	55.1	51.1	43.6
	H_2O-8x	21.3	37.8	42.1	46.6	36.9	21.5	23.7	21.1	21.9	74.0	90.5	42.9	65.5	54.6	50.8	43.4
	LLMLingua2-2x	8.0	39.1	41.0	42.5	33.9	18.1	25.8	19.8	26.6	15.5	63.9	36.5	68.0	25.8	37.9	33.5
	LLMLingua2-4x	13.0	33.2	33.3	43.8	24.2	24.4	25.3	22.4	24.7	4.4	79.1	34.4	22.5	19.9	44.6	29.9
	LLMLingua2-6x	17.1	34.0	26.2	40.2	20.4	18.5	25.0	21.7	23.6	2.8	76.8	34.1	18.0	17.5	45.2	28.1
	LLMLingua2-8x	19.8	28.9	23.4	35.2	23.4	17.5	24.1	21.6	22.9	0.0	76.1	34.6	13.0	16.1	47.7	26.9

Table 7: Full report of different compression methods on mistralai/Mistral-7B-Instruct-v0.2 across 15 datasets in LongBench

LLM	Method \ Dataset	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code	Avg.
		NarrativeQA	Qasper	MultiFieldQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PassageRetrieval	LCC	RepoBench-P	
Mistral-7B-Instruct-v0.2	Baseline	21.0	29.4	47.1	36.4	21.9	19.1	32.5	24.2	27.1	71.0	86.2	43.0	89.3	55.1	53.0	43.8
	KIVI-2bit	20.6	28.4	44.9	35.5	20.7	17.9	32.5	23.5	26.7	71.0	86.0	43.5	80.8	54.7	52.8	42.6
	KIVI-4bit	21.0	29.5	46.6	36.2	21.7	19.6	32.9	24.0	26.9	71.0	86.2	43.4	89.4	54.9	53.0	43.8
	FlexGen-4bit	22.2	29.9	47.0	34.8	21.6	16.9	32.4	24.0	26.9	69.5	86.4	42.6	83.0	54.4	53.0	43.0
	InfLLM-2x	21.6	24.2	46.1	35.0	20.9	18.3	31.0	23.4	25.9	67.5	86.7	41.2	65.8	54.8	53.6	41.1
	InfLLM-4x	20.9	16.8	38.4	33.9	19.2	18.2	29.6	22.2	24.7	60.5	88.3	41.3	41.4	52.8	55.3	37.5
	InfLLM-6x	19.9	14.6	36.9	31.8	16.4	14.7	29.1	22.1	23.8	57.0	87.4	40.4	32.6	53.2	53.6	35.6
	InfLLM-8x	20.9	12.8	33.0	29.1	16.2	13.3	27.9	21.2	23.8	60.0	86.0	40.1	26.2	54.1	53.5	34.5
	StreamLLM-2x	20.7	20.6	32.6	32.3	19.0	14.7	29.9	21.6	24.4	66.5	87.0	40.0	47.1	52.3	53.7	37.5
	StreamLLM-4x	19.7	15.1	25.4	27.7	17.4	14.6	27.4	20.2	22.1	61.0	83.7	39.2	31.6	51.8	55.9	34.2
	StreamLLM-6x	17.8	12.8	24.1	24.7	13.2	10.0	25.4	20.3	20.5	59.0	81.8	38.0	25.3	52.9	56.8	32.2
	StreamLLM-8x	16.8	11.3	22.9	22.8	12.0	10.7	24.6	19.8	19.7	56.5	79.6	38.8	16.9	53.8	56.0	30.8
	H_2O-2x	21.4	27.4	47.0	36.1	20.8	19.3	31.2	23.5	25.8	71.0	86.2	43.2	87.7	54.7	52.9	43.2
	H_2O-4x	21.6	24.9	44.8	35.0	19.0	17.5	28.6	22.8	24.2	71.0	86.7	43.8	82.9	53.9	52.3	41.9
	H_2O-6x	21.5	22.7	42.9	34.5	17.2	16.6	27.1	22.5	23.2	71.0	86.4	43.5	82.0	53.1	51.9	41.1
	H_2O-8x	20.9	21.4	41.1	32.8	16.8	15.9	26.2	22.6	23.0	71.0	86.3	43.8	79.5	52.8	51.8	40.4
	LLMLingua2-2x	20.2	26.8	38.9	34.7	16.8	17.7	29.9	23.6	25.8	19.0	81.2	36.5	54.9	21.9	41.5	32.6
	LLMLingua2-4x	18.0	22.1	35.0	31.6	16.4	15.9	26.9	22.7	24.1	3.5	80.0	34.0	14.0	18.9	47.3	27.4
	LLMLingua2-6x	15.7	18.4	29.5	25.7	15.5	11.0	26.0	21.2	22.7	2.0	80.7	34.0	8.9	19.1	50.3	25.4
	LLMLingua2-8x	15.3	16.7	26.9	23.9	15.1	8.9	25.2	21.4	22.1	0.5	81.5	33.5	8.0	19.3	51.7	24.7

Table 8: Full report of different compression methods on lmsys/longchat-7b-v1.5-32k across 15 datasets in LongBench

LLM	Method \ Dataset	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code	Avg.
		NarrativeQA	Qasper	MultiFieldQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PassageRetrieval	LCC	RepoBench-P	
LongChat-7b-v1.5-32K	Baseline	20.9	29.4	43.1	33.0	24.1	14.7	30.8	22.8	26.6	66.5	84.0	40.9	30.5	52.9	56.8	38.5
	KIVI-2bit	20.9	29.0	41.0	32.8	22.8	13.7	30.7	22.4	26.4	66.5	83.2	41.2	32.2	52.4	55.4	38.0
	KIVI-4bit	21.0	28.9	43.3	33.1	24.9	14.7	31.1	22.7	26.5	67.0	83.9	40.8	31.5	52.2	56.3	38.5
	FlexGen-4bit	20.5	30.4	43.2	33.7	23.8	13.9	31.7	22.9	26.5	66.0	81.5	40.9	31.5	50.4	56.3	38.2
	InfLLM-2x	19.1	28.4	40.0	30.1	26.6	13.2	31.1	21.7	24.6	60.0	84.1	11.3	19.5	48.9	54.9	34.2
	InfLLM-4x	17.6	21.2	33.9	32.4	25.4	14.4	29.4	21.5	22.3	55.0	84.3	9.9	9.5	41.4	52.1	31.4
	InfLLM-6x	16.5	17.0	29.8	30.6	24.6	15.2	28.4	21.0	21.1	55.5	82.4	8.7	10.0	41.4	51.6	30.2
	InfLLM-8x	14.9	16.5	29.1	26.4	23.1	15.4	26.6	20.7	20.5	48.5	78.8	8.7	6.0	41.6	50.8	28.5
	StreamLLM-2x	18.8	26.3	26.6	29.1	24.8	11.5	28.5	20.8	23.3	61.0	82.9	9.6	25.0	43.9	54.6	32.5
	StreamLLM-4x	18.1	19.1	22.4	29.5	25.1	11.6	25.5	20.9	20.4	54.5	81.1	11.4	13.5	49.7	54.2	30.5
	StreamLLM-6x	17.7	17.7	23.3	26.0	26.8	11.5	23.4	20.4	18.2	54.5	78.1	9.9	10.5	49.5	53.4	29.4
	StreamLLM-8x	13.9	16.7	21.9	24.3	26.4	13.6	21.6	19.8	17.1	49.0	73.9	10.4	10.0	52.4	51.4	28.2
	H_2O-2x	20.9	27.1	35.0	30.8	22.6	12.8	28.0	21.9	24.0	66.0	82.1	39.8	30.5	59.5	56.0	37.1
	H_2O-4x	21.4	25.3	32.1	30.6	22.9	12.2	23.0	21.7	21.1	65.5	80.6	39.7	28.5	56.2	54.3	35.7
	H_2O-6x	21.1	23.7	32.1	29.7	21.6	12.7	21.5	21.4	19.9	65.5	81.0	39.7	28.0	53.2	53.4	35.0
	H_2O-8x	20.4	22.4	32.7	29.3	21.2	12.3	20.1	20.9	18.5	65.5	80.5	38.8	28.5	50.2	52.7	34.3
	LLMLingua2-2x	13.3	27.5	36.3	28.5	25.4	13.0	28.5	22.3	25.6	6.0	65.3	34.8	19.5	16.1	49.0	27.4
	LLMLingua2-4x	14.4	26.3	30.7	27.2	24.0	10.7	25.1	22.0	23.4	1.0	61.9	32.0	5.5	16.0	47.7	24.5
	LLMLingua2-6x	14.7	25.6	27.6	24.3	24.7	11.7	23.8	21.6	22.4	0.0	64.4	32.6	5.0	15.2	48.6	24.1
	LLMLingua2-8x	14.6	24.4	24.9	23.8	23.5	11.2	23.1	21.4	21.4	0.5	66.6	31.5	6.5	16.7	48.3	23.9

Table 9: Full report of linear-time sequence models and mixed architecture across 15 datasets in LongBench

LLM	Method \ Dataset	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code	Avg.
		NarrativeQA	Qasper	MultiFieldQA	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PassageRetrieval	LCC	RepoBench-P	
Mamba	Mamba-2.8B	2.7	5.8	13.3	6.2	9.2	3.6	17.9	16.6	22.9	50.0	54.4	12.5	1.2	50.6	44.5	20.8
	Mamba-Chat-2.8B	3.2	6.3	17.9	7.3	9.5	4.0	21.5	18.5	23.5	45.5	43.2	23.9	3.7	50.5	45.0	21.6
	Mamba2-2.7B	2.6	5.6	14.4	7.9	8.7	3.5	20.3	17.8	24.8	45.0	58.2	18.4	4.1	54.4	45.5	22.1
RWKV	RWKV-5-World-7B	3.3	9.8	16.2	6.5	7.6	2.2	21.5	16.2	17.9	61.0	77.2	18.9	4.5	36.2	31.8	22.1
R-Gemma	R-Gemma-2B-it	12.0	16.2	26.0	9.8	10.8	4.3	20.7	20.0	22.1	52.0	63.3	23.6	4.0	57.0	50.3	26.1
	R-Gemma-9B-it	15.4	25.8	32.3	25.4	27.3	13.0	24.6	18.1	23.0	60.5	70.5	32.3	9.0	64.1	57.5	33.2