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

Anonymous ACL submission

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 over 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.

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. 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 Claude 3 offers a context length of 200K tokens.[1]

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 requires a KV cache of 3TB, 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 0.91GB, whereas, for a 100K context, it is 22.8GB (Fu, 2024). Given the limited memory space available for serving the model, supporting longer contexts usually reduces the number of requests that can be processed, leading to lower hardware utilization and, consequently, 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 unnecessary tokens to maintain a constant KV cache size (Xiao et al., 2023; Zhang et al., 2024c), compressing prompt into a shorter input (Jiang et al., 2023b; 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). How-

[1] https://www.anthropic.com/news/claude-3-family

Table 1: Evaluated 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 the fast serving. "N/A" means it can be directly accelerated by existing infrastructure.

Method	Taxonomy	KV Cache Complexity	Sys. Supports?
Mamba (Gu and Dao, 2023) RWKV (Peng et al., 2023)	Linear-time model	KV cache free	Yes Yes
RecurrentGemma (Botev et al., 2024)	Linear-time model + local attention	Constant	Yes
StreamingLLM (Xiao et al., 2023) H_2O (Zhang et al., 2024c) InfLLM (Xiao et al., 2024)	Token Dropping	Constant	Yes No Yes
LLMLingua (Jiang et al., 2023b)	Prompt Compression	Constant	N/A
KIVI (Liu et al., 2024b) FlexGen-4bit (Sheng et al., 2023)	Quantization	Linear	Yes Yes

ever, 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**[2] (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 range of modern long-context compression methods from various schools of thought. It assesses these methods against multiple tasks, each 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. Last, 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 will opensource 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

Before going into the experiment details, we provide brief introductions of 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 linearly with sequence length. Regarding system support scenarios, to the best of our knowledge, most methods have varying levels of system-level optimization, where 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), RWKV (Peng et al., 2023), HGRN (Qin et al., 2024), MEGA (Ma et al.,

[2] 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.

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:

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

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 the 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). Furthermore, based on these findings, the latest research has pushed quantization to 1bit (Zhang et al., 2024a; Zandieh et al., 2024).

2.3 Token Dropping

Based on the observation that attention scores are highly sparse, token dropping based methods drop the unimportant token from the KV cache (Zhang et al., 2024c; Xiao et al., 2023, 2024; Li et al., 2024b; Liu et al., 2024a). The transformer-based LLM inference workflow involves two stages: i) the *prefill stage*, where the input prompt is used to generate KV cache and the first output token; and ii) the *decoding stage*, where the model uses and updates KV cache to generate the next token one by one. **Token dropping-based methods fall into two main categories: dropping tokens during prefill or dropping tokens after prefill.** Dropping tokens during prefill means 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. **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 most token-dropping methods rely on accurate attention scores to determine token importance, which requires generating the full KV cache first. We closely follow the official or endorsed implementation of each method. In our benchmark, methods that drop tokens during prefill include StreamingLLM (Xiao et al., 2023) and InfLLM (Xiao et al., 2024). Methods that drop tokens after prefill include H_2O (Zhang et al., 2024c).

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 model-or-even-task-specific, requiring training tailored to specific LLMs and limiting their adaptability. In this work, we focus on general compression methods for fair comparison with other KV-cache compression approaches.

Natural Language Prompt Compression LLMLingua enhances LLM performance on long context tasks by converting them into short context tasks using coarse-to-fine prompt compression (Jiang et al., 2023b). It employs a budget controller to allocate compression ratios to different prompt parts dynamically, ensuring semantic integrity. Unlike LLMLingua's general approach, Nano-Capsulator (Chuang et al., 2024) provides task-specific compression to preserve long prompt performance better. In this study, we examine general compression methods for a fair comparison with other KV-cache compression techniques, using LLMLingua as a benchmark.

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.

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 the following: (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 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 follow the spirit of the technical report of Gemini 1.5 (Reid et al., 2024), but some small adjustments were made to ensure accommodate some community observations as well as better fairness. We refer our readers to Appendix A for further details.

For models, we elect to cover **3 representative transformer-based LLMs and 3 linear-time sequence model families**. For transformer-based LLMs, we opt to have Mistral-7b-Instruct-v0.2 (Jiang et al., 2023a), Longchat-7B-v1.5-32K (Li et al., 2023a) and Llama-8B-Instruct (AI@Meta, 2024) to provide a coverage of SOTA long-context capable model as well as the most recent progress of open sourced LLMs. For linear-time sequence models, we evaluated Mamba-Chat-2.8B (Gu and Dao, 2023), RWKV-5-World-7B-v2 (Peng et al., 2023), and RecurrentGemma-2b-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. For linear time sequence models and mixed architecture, we choose Mamba-2.8B (Gu and Dao, 2023), Mamba-chat (Mattern and Hohr, 2023), RWKV-5-World-7B-v2 (Peng et al., 2023), RecurrentGemma-2b-It and RecurrentGemma-9b-It (Botev et al., 2024). 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., 2024c), and InfLLM (Xiao et al., 2024). For *Prompt Compression*, we adopt LLMLingua (Jiang et al., 2023b). The hyperparameter setting for each method can be found in Appendix B.

3.2 Experiment Setup and Report Digestion

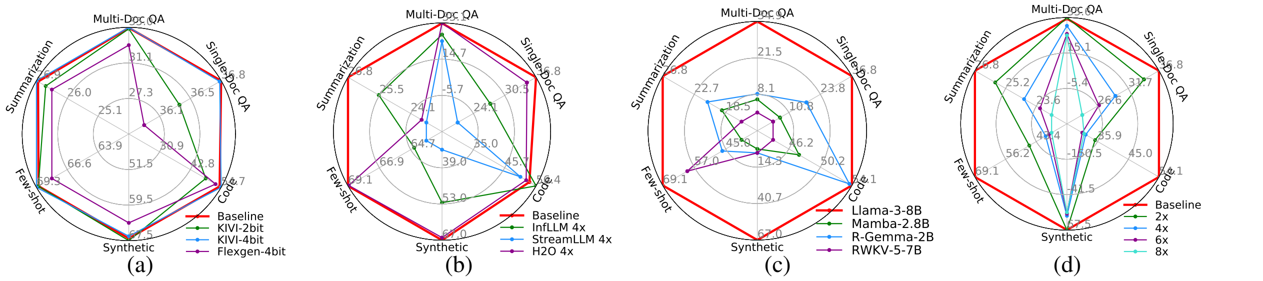
Given the vastly different design principles employed in different schools of long context handling methods, it is, in fact, impossible to achieve a global alignment where all covered methods are considered fairly aligned against each other. For example, while KV cache quantization methods like FlexGen (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 ap-

Table 2: Performance of KV cache quantization, token eviction, prompt compression, RNNs, and RNN-transformer hybrid methods on our benchmark. "Comp. Ratio" refers to the theoretical compression ratio, and "LB Avg." refers to average performance on LongBench.

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	34.9	26.8	69.1	67.0	54.1	45.2	100.0
	KIVI-2bit	5.05×	36.4	34.8	26.6	69.1	67.5	48.8	44.4	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×	35.9	33.0	26.4	67.9	63.5	52.6	43.9	100.0
	InfLLM-2x	2.00×	28.4	33.9	25.1	67.5	67.5	54.2	42.7	42.0
	InfLLM-4x	4.00×	27.5	28.6	25.5	64.4	52.5	56.4	40.2	42.0
	InfLLM-6x	6.00×	25.6	25.1	24.9	62.7	42.0	58.6	38.2	45.7
	InfLLM-8x	8.00×	23.5	25.0	24.6	62.5	34.0	59.4	37.3	37.3
	StreamLLM-2x	2.00×	23.9	31.4	24.8	67.7	50.0	46.0	39.0	24.0
	StreamLLM-4x	4.00×	20.9	24.9	23.4	63.6	32.0	51.1	35.5	25.0
	StreamLLM-6x	6.00×	17.9	20.2	22.4	60.3	24.0	54.9	33.1	23.0
	StreamLLM-8x	8.00×	16.5	18.3	21.1	58.8	18.5	55.2	31.6	22.3
	H_2O-2x	2.00×	35.9	34.8	25.4	69.1	66.5	54.3	44.7	30.0
	H_2O-4x	4.00×	35.0	35.1	23.7	69.0	66.0	53.0	44.0	30.0
	H_2O-6x	6.00×	33.8	35.1	22.7	69.0	66.0	53.2	43.6	30.0
	H_2O-8x	8.00×	33.7	35.0	22.2	69.1	65.5	52.7	43.4	30.0
	LLMLingua-2x	2.00×	34.3	35.6	25.8	46.3	67.5	35.2	37.6	51.3
	LLMLingua-4x	4.00×	29.6	30.8	24.3	39.4	23.5	32.4	30.7	8.3
	LLMLingua-6x	6.00×	26.8	26.1	23.4	37.9	17.0	31.3	28.2	0.7
	LLMLingua-8x	8.00×	24.0	25.3	22.8	36.9	13.0	31.8	26.9	0.0
Mistral-7B-Instruct-v0.2	Baseline	1.00×	32.5	25.8	27.9	66.8	89.3	52.4	43.5	100.0
	KIVI-2bit	5.05×	31.4	24.7	27.6	66.8	80.8	52.1	42.4	99.0
	KIVI-4bit	3.11×	32.3	25.8	28.0	66.9	89.4	52.4	43.5	99.0
	FlexGen-4bit	3.20×	31.7	25.1	27.6	65.9	82.3	52.4	42.5	98.3
	InfLLM-2x	2.00×	30.7	24.7	26.7	65.1	65.8	51.5	40.7	64.3
	InfLLM-4x	4.00×	25.5	23.8	25.6	63.2	42.4	51.5	37.3	31.6
	InfLLM-6x	6.00×	23.9	21.0	24.9	61.4	32.4	50.7	35.2	32.0
	InfLLM-8x	8.00×	22.6	20.3	24.4	61.2	23.9	50.3	34.0	28.3
	StreamLLM-2x	2.00×	24.3	22.1	25.3	64.6	47.1	50.9	37.2	54.7
	StreamLLM-4x	4.00×	20.4	19.9	23.3	61.2	31.6	50.8	33.8	32.0
	StreamLLM-6x	6.00×	18.4	16.0	22.1	59.7	25.3	52.1	31.9	25.0
	StreamLLM-8x	8.00×	17.3	15.2	21.4	58.7	16.9	52.5	30.6	19.3
	H_2O-2x	2.00×	35.7	29.7	26.7	66.8	84.8	53.8	44.6	97.3
	H_2O-4x	4.00×	34.3	28.7	24.9	67.2	83.5	53.1	43.7	93.3
	H_2O-6x	6.00×	33.7	28.2	24.3	66.9	82.7	52.5	43.2	86.0
	H_2O-8x	8.00×	32.8	27.6	23.6	67.0	84.2	52.3	42.8	79.7
	LLMLingua-2x	2.00×	28.4	23.0	26.5	45.3	54.9	30.9	32.4	42.0
	LLMLingua-4x	4.00×	25.1	21.3	24.6	39.0	14.0	32.0	27.2	10.7
	LLMLingua-6x	6.00×	21.2	17.4	23.3	38.6	8.9	33.3	25.1	0.3
	LLMLingua-8x	8.00×	19.6	16.1	22.9	38.0	8.0	34.0	24.4	0.0
Mamba	Mamba-2.8B	-	7.2	6.3	19.1	38.9	1.2	47.5	20.7	10.7
	Mamba-Chat-2.8B	-	2.0	4.0	1.4	11.5	0.0	20.7	6.6	0.0
RWKV	RWKV-5-World-7B	-	4.3	1.5	16.5	59.7	4.0	44.3	22.6	4.3
R-Gemma	R-Gemma-2B-it	-	18.1	8.3	20.9	46.3	4.0	53.8	26.2	23.3
	R-Gemma-9B-it	-	24.5	21.9	21.9	54.4	9.0	60.6	33.2	27.0

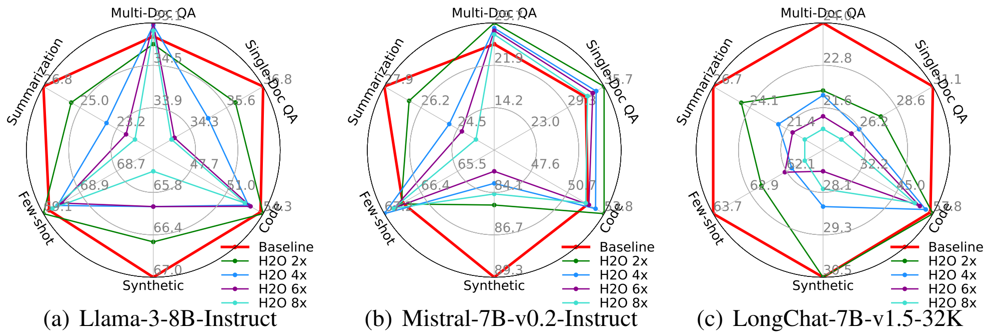
proaches typically employ a constant size of kept tokens and evict everything else, making their compression gain dynamic against inputs of different lengths; again, they are not aligned with the KV cache quantization method nor KV cache-free approaches. Note, the abovemented are merely alignment hardship due to conflict of long context handling schools, 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, 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-sourced datasets, their architecture compared to models like Llama-3 — which are pre-trained upon proprietary data corpus and done so with an overtrained recipe that has proven to be beneficiary.

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



(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.

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 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.

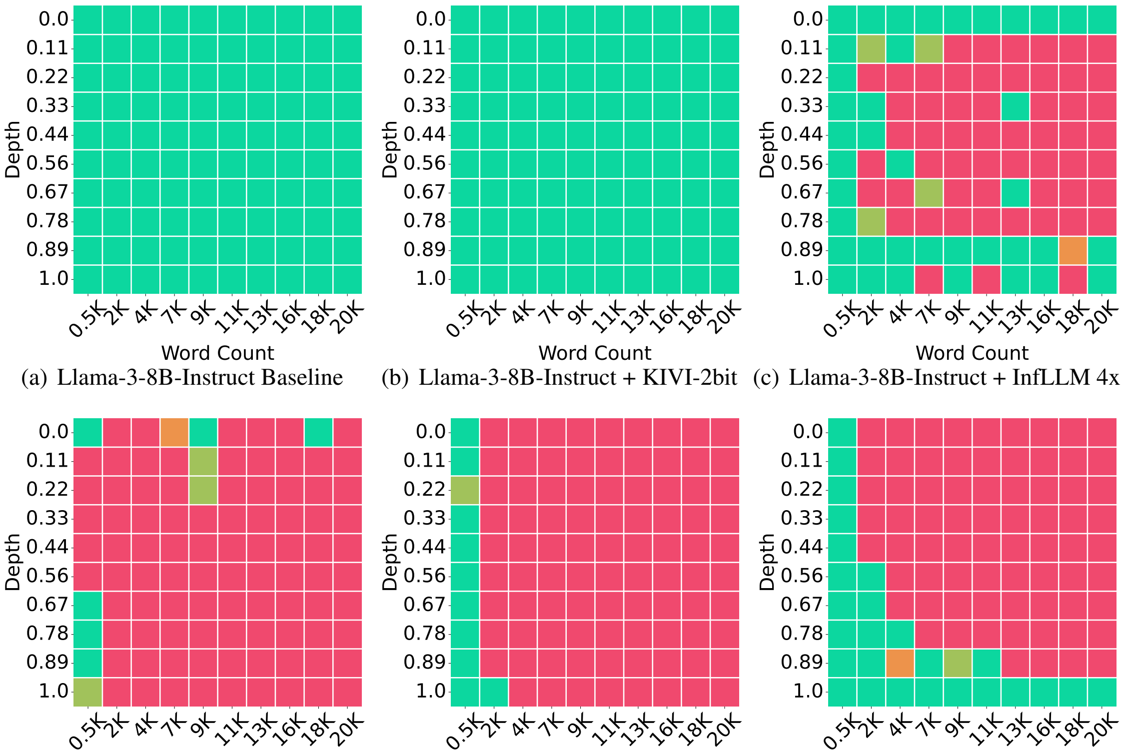
With such efforts, our experiment report should be reasonably comparable among similar compression ratios. Though we note our additional alignment effort will not resolve the pretraining difference among different backbone models — where an aligned comparison here can only be done by training different 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 opensourced models like Llama (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 Llama-3-8B (AI@Meta, 2024), as well as several other covered linear and mixed models. Based on all of the 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 representation 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., 2024c) do not employ any compression operation during the prefill stage, which often leads to much better results than methods which do compress within the prefill stage, namely StreamingLLM (Xiao et al., 2023), InfLLM (Xiao et al., 2024), and LLMLingua (Jiang et al., 2023b).

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 (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



(a) Llama-3-8B-Instruct Baseline (b) Llama-3-8B-Instruct + KIVI-2bit (c) Llama-3-8B-Instruct + InfLLM 4x

(d) LLMLingua-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 Architecture. The best method in each school of approaches is featured with comparable compression ratios.

naturally carry more influence than compression during the decoding stage. More on this in Section 6.

OB ❷ Carefully designed 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 evicting task-influential tokens by accident (e.g., one can imagine evicting 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 its memory footprint *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., 2024c) tend to perform exceptionally well on code-related tasks, with Figure 2 and Figure 22 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 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.

Conversely, hard prompt compression methods like LLMLingua (Jiang et al., 2023b) perform the worst on the needle test across all KV cache-required methods — which is, once again, a well-expected finding as if one evicted the needle information within the input, the LLM will certainly not be able to answer the retrieval-required question correctly. LLMLingua 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 upon 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 acceptable footprint increase.

OB ❹ Needle-in-a-haystack test remains challenging for constant KV cache/KV cache-free 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), LLMLingua by Jiang et al. (2023b) (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 constant KV cache or KV cache-free methods often struggle to maintain good retrieval performance as the baseline methods. While we believe different architectural 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 finetuned version of Mamba-2.8B (Gu and Dao, 2023) with long context focuses — tend to have much better needle performance.

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 must consider this challenge when developing efficient yet long context-capable methods with real-world efficiency in mind.

5 Conclusion

Our benchmark fills a critical gap by providing a detailed and accessible pipeline to evaluate various long context-capable approaches across a wide range of long context tasks. We offer a comprehensive evaluation of 11 methods under 65 settings, which set the empirical foundation for unmasking many previously unknown phenomena and insights. Outside the empirical and analytical novelties we present, our contributions also extend to providing a minimalistic, reproducible, yet extensible benchmarking package to all interesting scholars.

6 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[3] 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.

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*.

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

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

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*.

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.

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*.

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*.

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*.

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, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *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, 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*.

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.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024a. 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. 2024b. 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. *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*.

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*.

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*.

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*.

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*.

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*.

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

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024a. 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, and Maosong Sun. 2024b. ∞bench: Extending long context evaluation beyond 100k tokens. *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. 2024c. 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 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 omit the results of PassageCount since, for counting tasks, 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.

A.2 Needle-in-a-Haystack

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.`", and Greg Kamradt 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."* 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 needle (gkamradt, Arize-ai, (Levy et al., 2024)) or passkey retrieval tasks (Reid et al., 2024; Hsieh et al., 2024) existing in the community, we clarify the formation of our needle task as the following:

```
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[4]>
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

We follow the settings in the LongBench official implementation, and following the settings for other models' configurations, we set the `max_length` parameter as in Table 3.

Table 3: `max_length` setting in LongBench.

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

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

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. `20480` usually converts to roughly 32k tokens with the tokenizer utilized in models like Mistral-7B-v0.2-Instruct (Jiang et al., 2023a).

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 θ (Su et al., 2024) setting to 32x of its original size due to its limited 8k off-the-shelf context window.

B.3 Hyperparameter Setting

Linear-time sequence models and mixed architecture In our paper, we benchmark five linear time sequence models. While linear time sequence 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 the Needle-in-a-Haystack task (Mohtashami and Jaggi, 2023; Hsieh et al., 2024), we uniformly set the context length to 32k 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
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, we use the official implementation [5], and for FlexGen, we follow the group-wise quantization in the official codebase [6]. The group size for both KIVI and FlexGen is set to 32. Furthermore, for KIVI, the residual length is set to 128.

Token Dropping We evaluate two popular token dropping methods used for handling long contexts: StreamLLM (Xiao et al., 2023) and H_2O (Zhang et al., 2024c). 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% to achieve compression gains of 2x, 4x, 6x, and 8x, respectively.

Prompt Compression We evaluate LLMLingua on four different compression rates, which are 2×, 4×, 6×, and 8×. K× denotes that the compressor are restricted to compress the length into 1/K of the original length of long inputs. In the two-needle dataset, we expanded the rope base numbers by 32 times to overcome the constraints imposed by Llama3-8B's limited 8k context window. This approach is necessary because even with a 2× compression rate on the two-needle dataset, the resulting size can still exceed 8K, potentially degrading the performance of Llama3-8B. All other experiment settings, including configurations on datasets and hyperparameters of LLMs, are all identical to other KV cache compression benchmarks.

[5] https://github.com/jy-yuan/KIVI
[6] https://github.com/FMInference/FlexGen

C Related Works

A few related benchmarking works also discuss the long context problem in LLMs. LongBench (Bai et al., 2023b) provides a bilingual, multitask benchmark for long context understanding. InfiniBench (Zhang et al., 2024b) extends the benchmark context length to 100k tokens, and LongICLBench (Li et al., 2024a) provides a more reliable benchmarking dataset closer to real-world scenarios. Another recent work, Ruler (Hsieh et al., 2024), focuses on finding the "real" context size of LLMs.

Unlike other works that mainly focus on producing datasets or benchmarking different models, our work presents comprehensive results primarily focusing on the **comparison between long context-capable approaches**, that covers 10+ long context-capable approaches under 60+ different settings.

D More Experimental Results

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

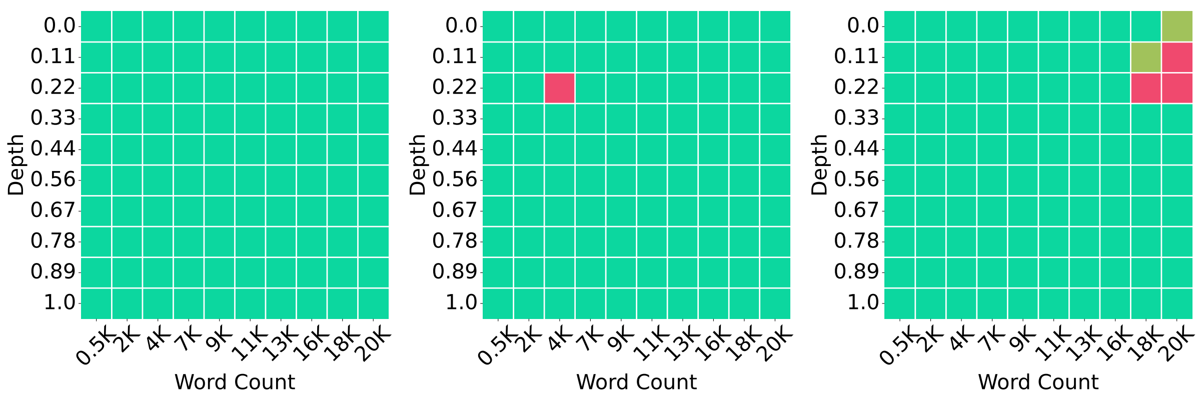
Table 5 shows all the LongChat-7B results on LongBench and the Needle experiment. We present FlexGen results on three different LLMs in Figure 6. Additional H_2O results for different compression ratios on Llama-3-8B, LongChat-7B-v1.5, and Mistral-7B-v0.2 can be found in Figure 13, 14 and 15 respectively.

We provide more visualization results on Needle task. For baseline performance for the three models in Figure 4. For InfLLM results on the LongChat and Mistral models, the results are listed in Figure 8 and 9. Figure 20 and 21 show the performance of quantization, token dropping, and prompt compression on Mistral and LongChat, respectively. Figure 22, 23 and 24 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.

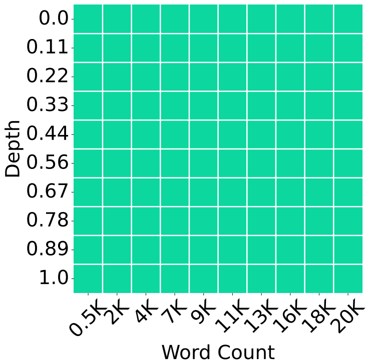
Table 5: Performance of KV cache quantization, token eviction, and prompt compression methods on LongChat-7B in our benchmark.

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	24.0	26.7	63.7	30.5	56.9	38.7	100.0
	KIVI-2bit	5.05×	30.2	23.2	26.4	63.7	32.3	55.9	38.3	85.6
	KIVI-4bit	3.11×	30.9	24.2	26.9	63.8	31.5	56.4	38.8	96.3
	FlexGen-4bit	3.20×	30.3	23.0	26.5	61.5	31.0	52.4	37.3	94.6
	InfLLM-2x	2.00×	11.5	4.3	13.2	10.1	0.1	23.6	11.0	5.3
	InfLLM-4x	4.00×	14.6	8.8	18.4	18.1	0.9	26.5	15.6	6.7
	InfLLM-6x	6.00×	15.6	13.1	20.1	23.9	1.3	27.5	18.3	0.1
	InfLLM-8x	8.00×	15.5	14.9	21.4	27.0	5.0	26.2	19.6	9.7
	StreamLLM-2x	2.00×	5.5	1.8	9.1	4.3	0.5	19.6	6.8	0.0
	StreamLLM-4x	4.00×	9.0	4.8	12.1	10.8	0.0	27.1	10.9	3.0
	StreamLLM-6x	6.00×	13.2	8.1	13.7	16.7	0.0	29.0	14.2	2.7
	StreamLLM-8x	8.00×	12.4	10.0	14.4	22.2	0.3	26.4	15.3	2.7
	H_2O-2x	2.00×	27.6	22.1	24.7	62.6	30.5	57.8	37.1	56.7
	H_2O-4x	4.00×	26.2	21.9	22.0	61.9	28.5	55.3	35.7	28.3
	H_2O-6x	6.00×	25.7	21.3	20.9	62.1	27.5	53.2	34.9	19.7
	H_2O-8x	8.00×	25.0	21.0	20.0	61.6	28.0	51.4	34.2	14.3
	LLMLingua-2x	2.00×	26.5	22.2	25.4	35.5	19.5	32.5	27.6	28.7
	LLMLingua-4x	4.00×	23.8	20.8	23.6	31.6	5.5	31.8	24.6	3.3
	LLMLingua-6x	6.00×	22.6	20.2	22.6	32.4	5.0	31.9	24.2	0.6
	LLMLingua-8x	8.00×	21.5	19.5	21.9	32.8	6.5	32.6	23.9	0.0

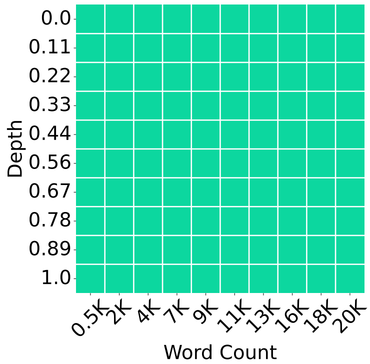


(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

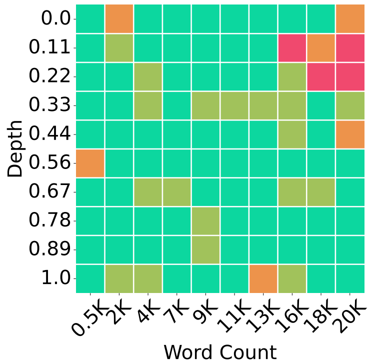
Figure 4: Baseline performance under needle test on three commonly used LLMs



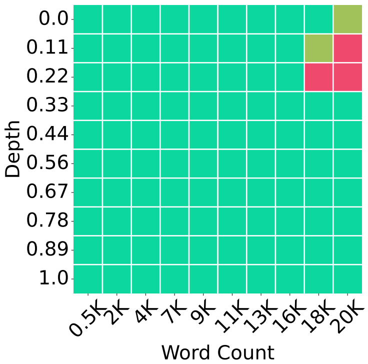
(a) Llama-3 + KIVI-2bit



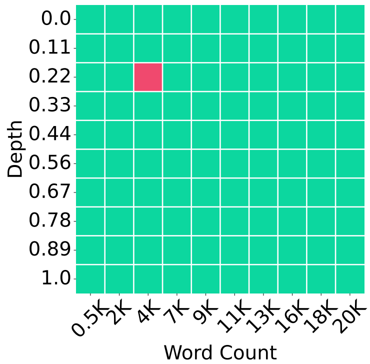
(b) Llama-3 + KIVI-4bit



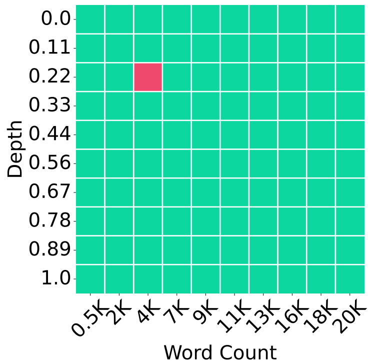
(c) LongChat-7B + KIVI-2bit



(d) LongChat-7B + KIVI-4bit

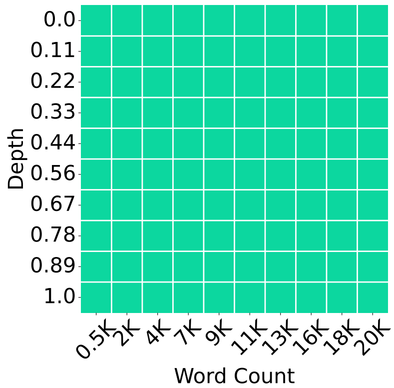


(e) Mistral-7B + KIVI-2bit

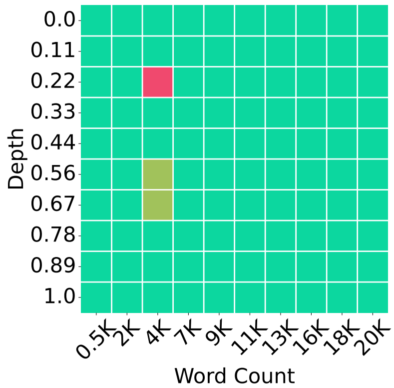


(f) Mistral-7B + KIVI-4bit

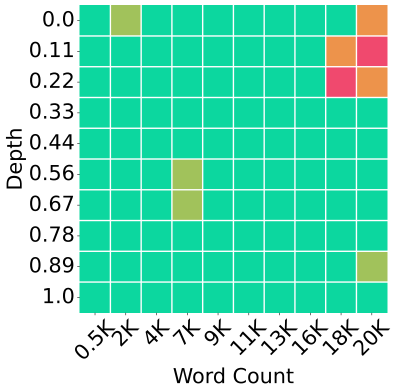
Figure 5: KIVI performance under needle test on three commonly used LLMs with 2-bit and 4-bit quantization



(a) Llama-3-8B-Instruct

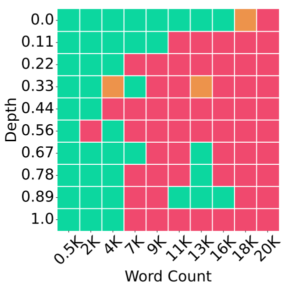


(b) Mistral-7B-v0.2-Instruct

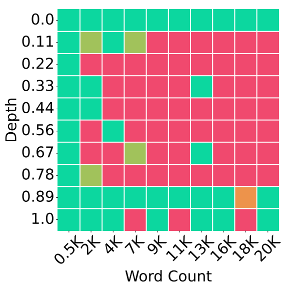


(c) LongChat-7B-v1.5-32K

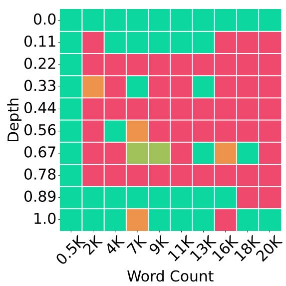
Figure 6: FlexGen performance under needle test on three commonly used LLMs



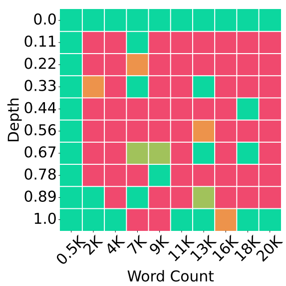
(a) 2x Compression



(b) 4x Compression

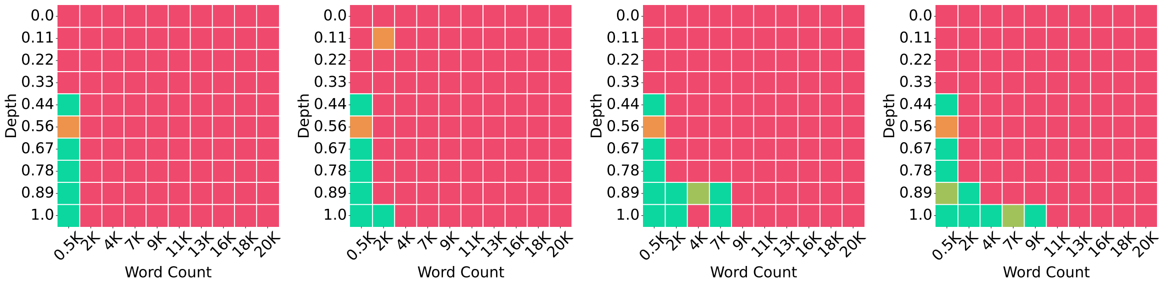


(c) 6x Compression



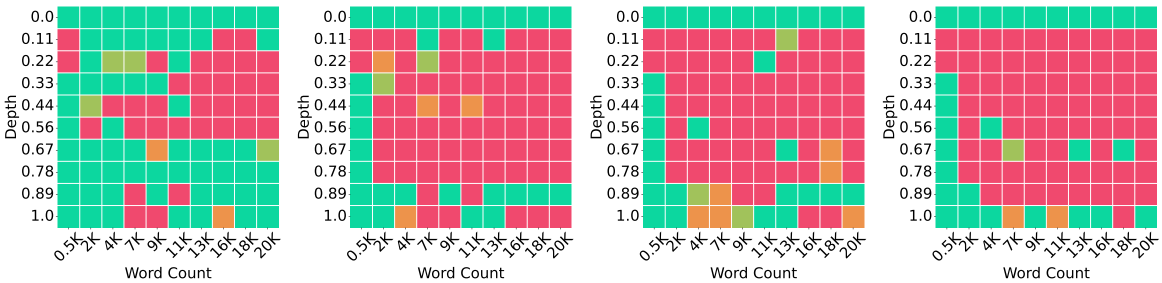
(d) 8x Compression

Figure 7: InfLLM on Llama-3-8B-Instruct with 4 different compression rate under needle test



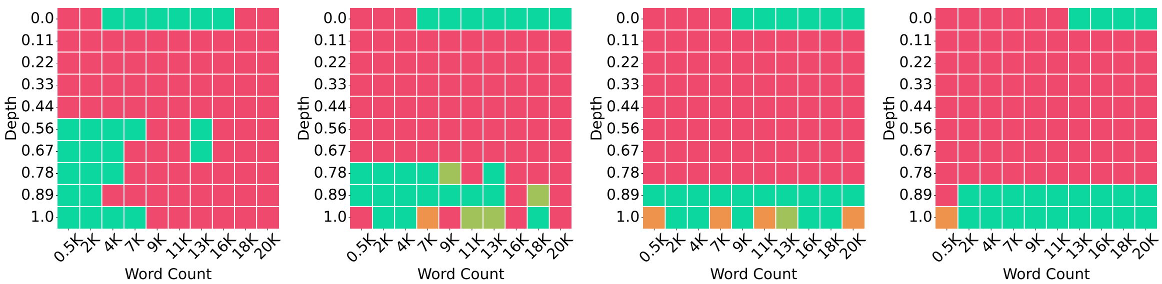
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 8: InfLLM on LongChat-7B-v1.5-32K with 4 different compression rate under needle test



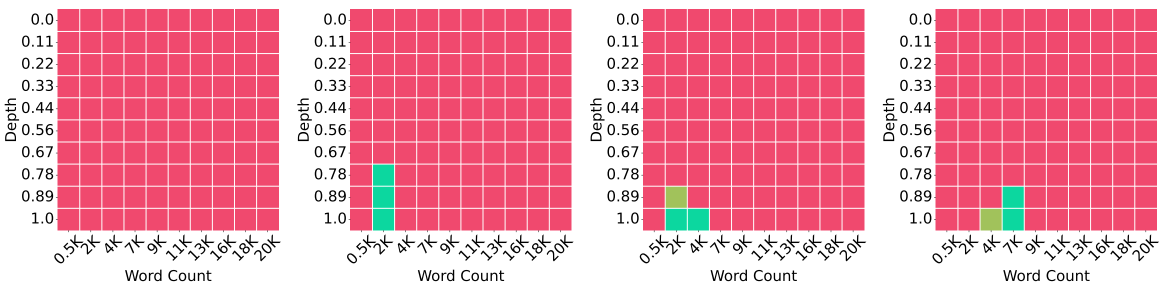
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 9: InfLLM on Mistral-7B-v0.2-Instruct with 4 different compression rate under needle test



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

Figure 10: StreamLLM on Llama-3-8B-Instruct with 4 different compression rate under needle test

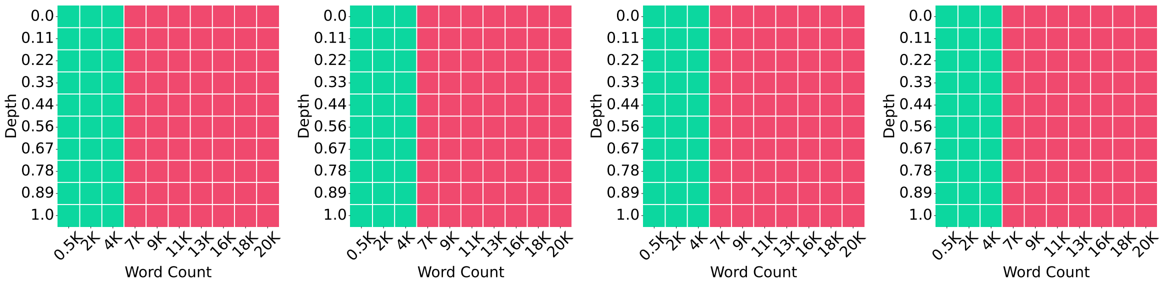


(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 11: StreamLLM on LongChat-7B-v1.5-32K with 4 different compression rate under needle test

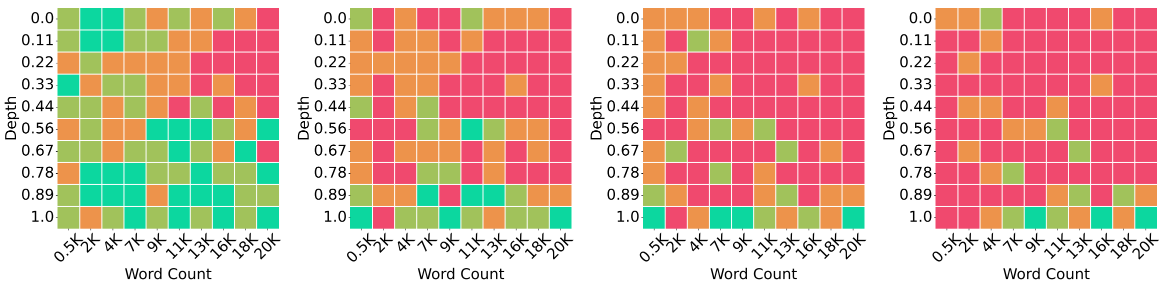
(a) 2x Compression (b) 4x Compression (c) 6x Compression (d) 8x Compression

Figure 12: StreamLLM on Mistral-7B-v0.2-Instruct with 4 different compression rate under needle test



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

Figure 13: H_2O on Llama-3-8B-Instruct with 4 different compression rate under needle test



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

Figure 14: H_2O on LongChat-7B-v1.5-32K with 4 different compression rate under needle test

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

Figure 15: H_2O on Mistral-7B-v0.2-Instruct with 4 different compression rate under needle test

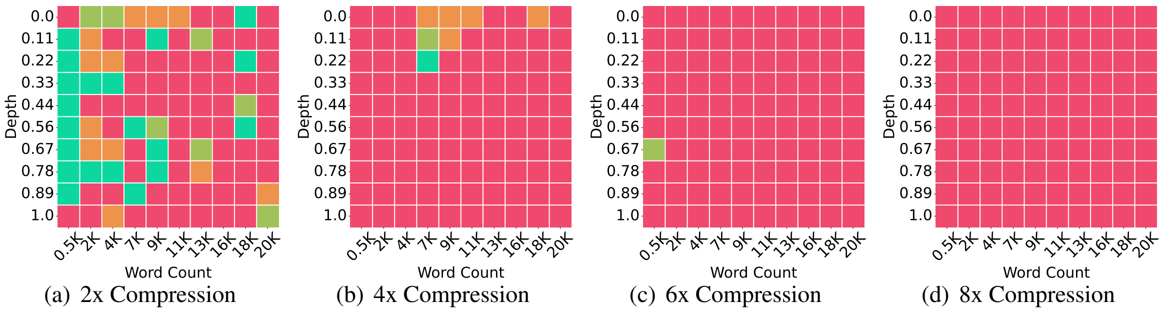


Figure 16: LLMLingua on LongChat-7B-v1.5-32K with 4 different compression rate under needle test

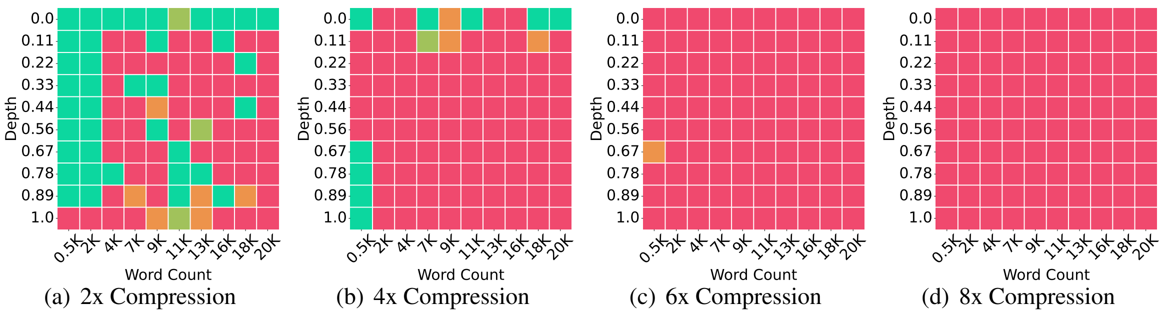


Figure 17: LLMLingua on Mistral-7B-v0.2-Instruct with 4 different compression rate under needle test

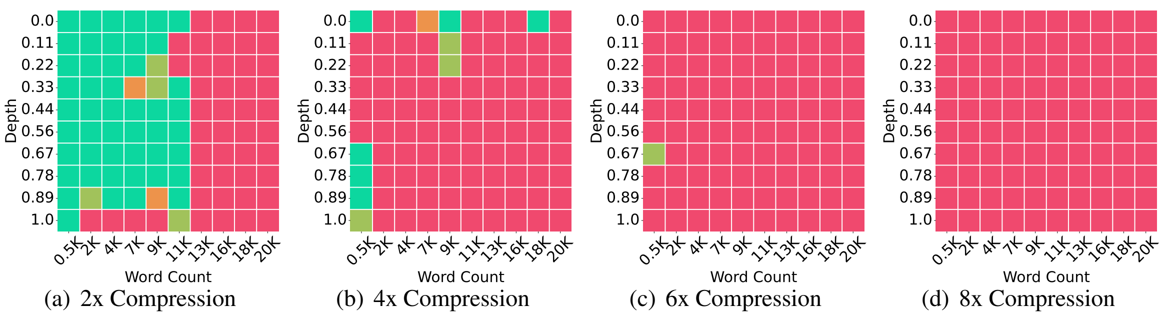
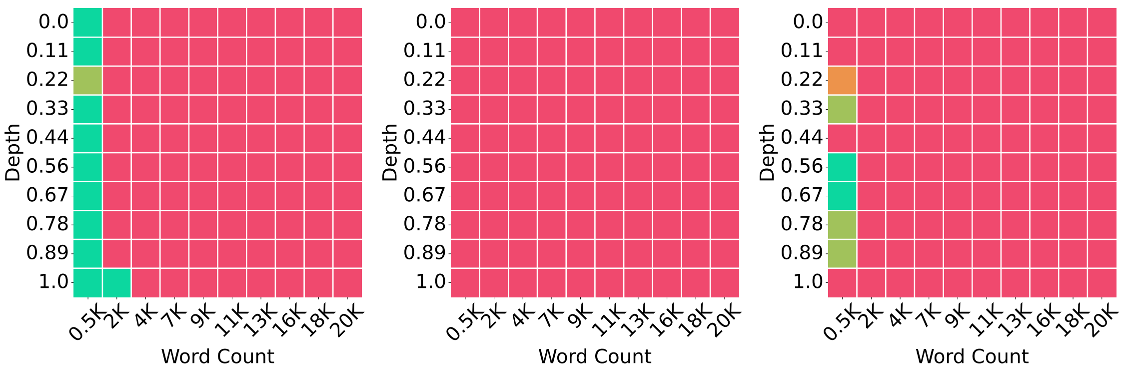


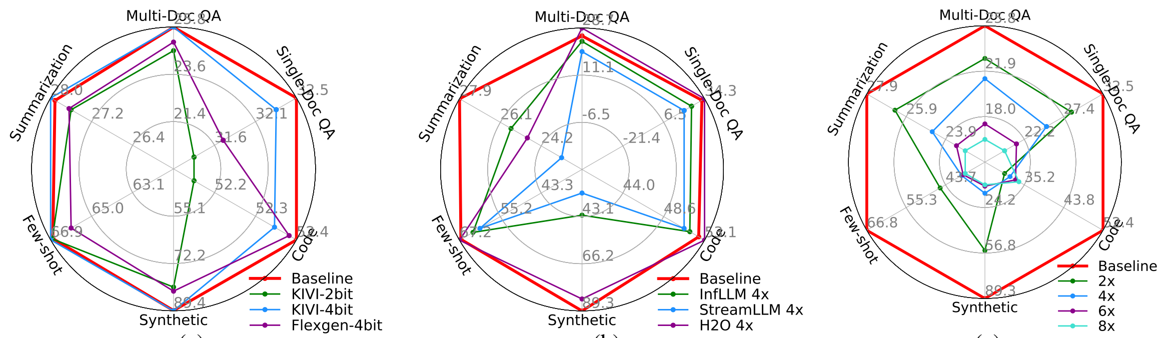
Figure 18: LLMLingua on Llama-3-8B-Instruct with 4 different compression rate under needle test



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

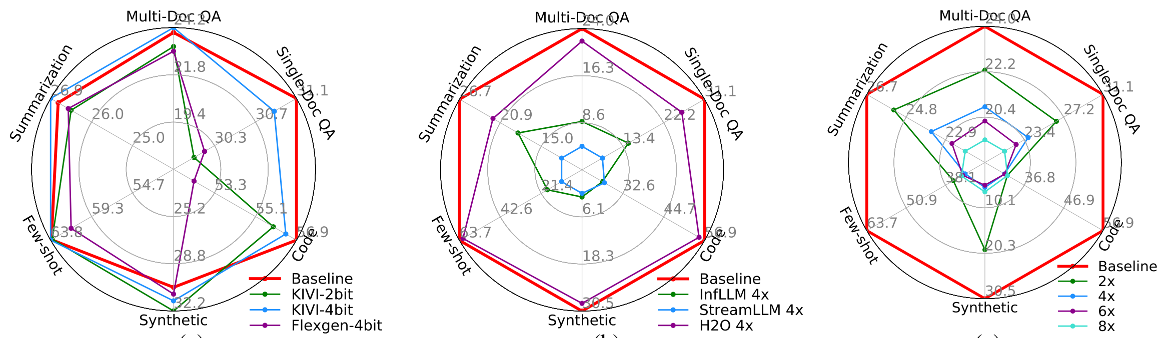
(d) RecurrentGemma-9B-it (e) RecurrentGemma-2B-it

Figure 19: Linear-time sequence models under needle test



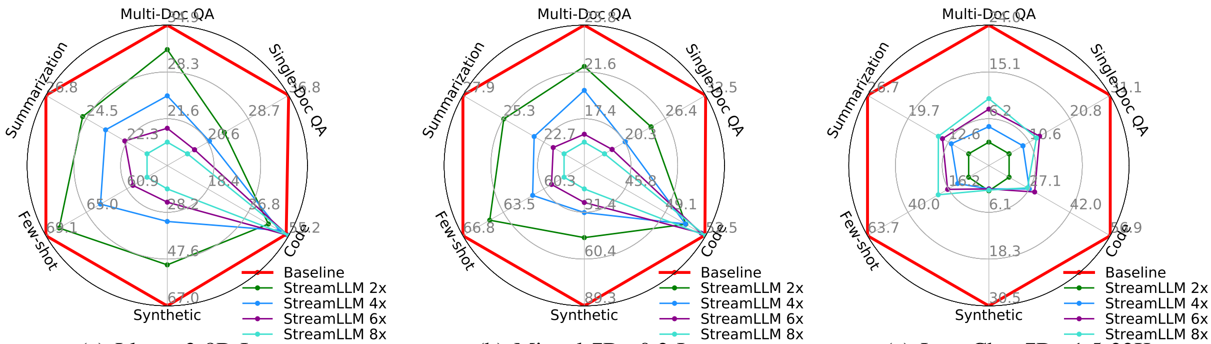
(a) (b) (c)

Figure 20: Mistral-7B-v0.2-Instruct with different compression methods (a) with Quantization; (b) with Token Dropping (c) with prompt compression.



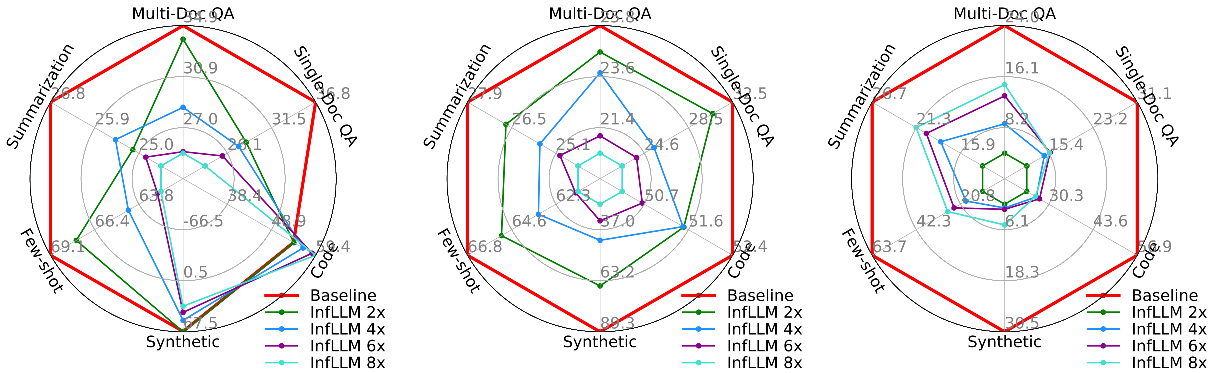
(a) (b) (c)

Figure 21: Longchat-7B-v1.5-32K with different compression methods (a) with Quantization; (b) with Token Dropping (c) with prompt compression.



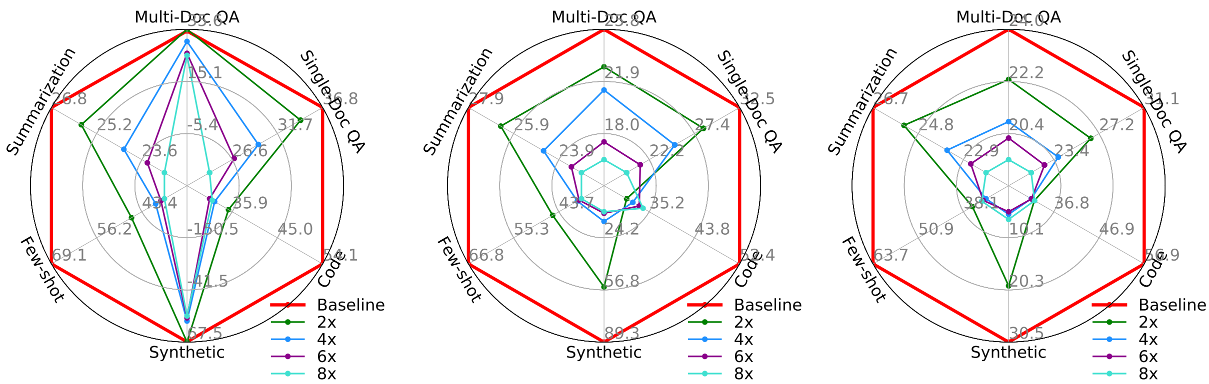
(a) Llama-3-8B-Instruct (b) Mistral-7B-v0.2-Instruct (c) LongChat-7B-v1.5-32K

Figure 22: 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 23: 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 24: LLMLingue with different compression ratios on three commonly used LLMs.

Table 6: Performance of different compression methods on Llama across all datasets in LongBench

LLM	Dataset / Method	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	44.2	44.5	46.8	36.4	21.5	30.0	22.7	27.8	74.5	90.2	42.5	67.0	57.0	51.2	45.2
	KIVI-2bit	21.4	43.2	44.5	46.8	37.1	20.6	30.0	22.1	27.8	74.5	90.5	42.3	67.5	50.8	46.7	44.4
	KIVI-4bit	21.0	44.8	44.6	47.0	36.5	21.4	30.2	22.4	28.0	74.5	90.3	43.0	66.5	57.4	52.0	45.3
	FlexGen-4bit	21.9	43.4	42.5	45.5	31.6	22.0	29.7	22.0	27.5	73.5	88.5	41.7	63.5	56.5	48.6	43.9
	InfLLM-2x	8.8	39.2	37.3	46.7	31.3	23.6	29.2	19.9	26.3	69.5	90.6	42.5	67.5	57.2	51.2	42.7
	InfLLM-4x	18.3	28.4	35.8	40.3	24.7	20.7	29.9	20.6	26.0	61.0	90.0	42.2	52.5	57.5	55.2	40.2
	InfLLM-6x	19.5	24.8	32.5	37.7	20.4	17.3	29.1	19.9	25.6	57.0	89.2	41.8	42.0	60.7	56.4	38.2
	InfLLM-8x	19.4	21.4	29.7	38.9	21.6	14.4	28.5	19.7	25.6	58.0	87.9	41.5	34.0	60.4	58.3	37.3
	StreamLLM-2x	9.3	34.0	28.5	42.4	29.9	22.0	29.0	19.9	25.4	71.0	90.3	41.9	50.0	47.7	44.3	39.0
	StreamLLM-4x	17.2	23.5	22.0	32.9	23.1	18.6	27.7	19.9	22.7	63.0	86.5	41.2	32.0	50.8	51.3	35.5
	StreamLLM-6x	17.1	18.8	17.9	27.9	19.3	13.5	26.8	19.2	21.2	58.0	82.0	40.9	24.0	57.1	52.6	33.1
	StreamLLM-8x	16.8	16.5	16.3	25.5	18.2	11.2	25.1	18.6	19.7	58.0	78.0	40.5	18.5	58.0	52.4	31.6
	H_2O-2x	21.5	42.7	43.5	46.4	36.5	21.5	28.2	22.1	26.1	74.0	90.6	42.8	66.5	57.1	51.6	44.7
	H_2O-4x	21.8	41.2	41.9	46.8	36.9	21.5	25.8	21.6	23.7	74.0	90.6	42.5	66.0	54.8	51.2	44.0
	H_2O-6x	21.5	38.3	41.7	46.8	36.8	21.7	24.6	21.1	22.4	74.0	90.5	42.5	66.0	55.4	51.0	43.6
	H_2O-8x	21.3	37.8	42.1	46.6	36.9	21.5	23.7	21.2	21.8	74.0	90.5	42.7	65.5	54.6	50.8	43.4
	LLMLingua-2x	22.0	40.0	41.0	46.9	33.9	25.9	28.2	22.5	26.6	15.5	86.6	36.8	67.5	25.9	44.4	37.6
	LLMLingua-4x	22.0	33.3	33.4	43.8	24.2	24.4	25.5	22.5	24.8	4.9	79.1	34.3	23.5	19.8	44.9	30.7
	LLMLingua-6x	19.9	34.2	26.2	40.2	20.2	18.0	24.9	21.7	23.7	2.8	76.7	34.2	17.0	17.5	45.1	28.2
	LLMLingua-8x	20.0	28.6	23.4	35.1	23.4	17.5	24.1	21.5	22.8	0.0	76.0	34.8	13.0	16.0	47.6	26.9

Table 7: Performance of different compression methods on Mistral across all datasets in LongBench

LLM	Dataset / Method	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	36.5	21.8	19.1	32.6	24.0	27.1	71.0	86.2	43.0	89.3	53.5	51.4	43.5
	KIVI-2bit	20.6	28.7	44.9	35.5	20.7	18.0	32.6	23.7	26.5	71.0	86.0	43.3	80.8	53.0	51.2	42.4
	KIVI-4bit	21.0	29.4	46.5	36.3	21.7	19.5	33.0	24.1	26.9	71.0	86.2	43.3	89.4	53.3	51.4	43.5
	FlexGen-4bit	20.2	28.6	46.3	35.9	20.9	18.5	32.4	23.6	26.9	69.0	85.2	43.6	82.3	52.5	52.4	42.5
	InfLLM-2x	22.0	24.2	46.0	35.1	20.9	18.0	30.9	23.1	26.0	67.0	86.9	41.3	65.8	52.4	50.6	40.7
	InfLLM-4x	21.0	17.0	38.5	33.3	19.1	18.9	29.9	21.9	24.9	60.5	87.9	41.3	42.4	50.3	52.7	37.3
	InfLLM-6x	20.2	14.4	37.0	31.9	15.9	15.3	28.6	21.8	24.4	56.0	87.4	40.7	32.4	50.0	51.3	35.2
	InfLLM-8x	20.1	12.8	34.9	29.6	17.1	14.2	28.2	20.9	24.0	58.5	85.2	40.0	23.9	49.9	50.6	34.0
	StreamLLM-2x	19.7	20.6	32.6	32.3	19.2	14.7	29.9	21.6	24.4	66.5	87.2	40.2	47.1	50.5	51.2	37.2
	StreamLLM-4x	20.7	15.1	25.4	27.7	17.4	14.7	27.6	20.3	22.1	61.0	83.7	38.9	31.6	49.2	52.5	33.8
	StreamLLM-6x	18.0	12.9	24.4	24.7	13.1	10.0	25.4	20.2	20.7	58.5	82.4	38.2	25.3	50.8	53.4	31.9
	StreamLLM-8x	17.5	11.3	22.9	23.0	12.0	10.7	24.8	19.8	19.7	57.0	80.6	38.4	16.9	51.4	53.6	30.6
	H_2O-2x	27.1	31.4	48.6	43.0	26.5	19.5	30.6	23.8	25.8	71.0	86.2	43.2	84.8	54.7	52.9	44.6
	H_2O-4x	26.6	28.6	47.8	41.9	26.0	18.4	27.4	23.4	23.8	71.0	86.7	43.8	83.5	54.0	52.2	43.7
	H_2O-6x	27.1	27.1	46.9	41.8	25.5	17.5	26.5	23.3	23.0	71.0	86.4	43.5	82.7	53.2	51.9	43.2
	H_2O-8x	26.6	25.7	46.1	41.0	24.8	16.9	25.4	22.6	22.8	71.0	86.3	43.6	84.2	52.7	51.8	42.8
	LLMLingua-2x	19.7	26.7	38.8	34.6	16.8	17.7	30.0	23.6	25.8	18.5	80.9	36.6	54.9	21.8	39.9	32.4
	LLMLingua-4x	18.1	22.2	35.0	31.6	16.4	15.9	26.9	22.7	24.1	3.5	79.9	33.8	14.0	19.2	44.9	27.2
	LLMLingua-6x	15.6	18.4	29.5	25.7	15.5	11.0	25.9	21.2	22.8	2.0	80.1	33.7	8.9	18.6	47.9	25.1
	LLMLingua-8x	15.2	16.7	27.0	23.8	15.1	9.2	25.3	21.2	22.1	0.5	80.3	33.2	8.0	18.6	49.3	24.4

Table 8: Performance of different compression methods on LongChat across all datasets in LongBench

LLM	Dataset / Method	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	20.7	29.4	43.2	33.1	24.1	14.7	30.9	22.8	26.6	66.5	84.0	40.8	30.5	54.8	58.9	38.7
	KIVI-2bit	20.8	28.7	41.0	32.9	23.0	13.8	30.5	22.6	26.3	66.5	83.2	41.3	32.3	54.1	57.6	38.3
	KIVI-4bit	20.5	28.9	43.2	33.1	24.9	14.7	31.4	22.8	26.5	67.0	83.9	40.6	31.5	54.1	58.8	38.8
	FlexGen-4bit	20.2	28.6	42.0	32.6	23.5	12.9	31.0	22.3	26.2	62.5	80.8	41.3	31.0	48.6	56.2	37.3
	InfLLM-2x	1.4	16.0	17.1	2.8	9.4	0.7	9.8	8.0	21.8	13.0	13.6	3.6	0.1	25.3	21.9	11.0
	InfLLM-4x	1.9	17.6	24.3	8.2	16.0	2.2	17.9	15.1	22.0	27.0	20.6	6.8	0.9	27.1	25.9	15.6
	InfLLM-6x	4.5	15.8	26.5	15.9	20.1	3.4	21.6	16.9	21.9	30.0	33.5	8.2	1.3	29.0	25.9	18.3
	InfLLM-8x	6.2	14.8	25.5	17.9	20.3	6.3	23.1	19.8	21.2	27.5	44.6	9.0	5.0	26.7	25.7	19.6
	StreamLLM-2x	0.9	6.7	8.8	1.2	4.0	0.3	4.1	5.5	17.6	5.0	5.6	2.4	0.5	21.1	18.1	6.8
	StreamLLM-4x	1.1	12.5	13.4	3.7	10.5	0.1	10.0	9.3	17.1	15.0	12.4	4.8	0.0	29.8	24.3	10.9
	StreamLLM-6x	2.1	17.2	20.3	7.3	15.8	1.1	14.4	13.6	13.0	24.8	20.0	5.3	0.0	29.8	28.3	14.2
	StreamLLM-8x	3.5	14.8	19.0	7.4	20.8	1.9	17.4	15.8	9.9	31.0	28.5	7.2	0.3	23.2	29.5	15.3
	H_2O-2x	20.7	27.2	35.0	30.8	22.6	12.8	28.4	21.8	23.9	66.0	82.1	39.8	30.5	59.5	56.0	37.1
	H_2O-4x	21.2	25.2	32.1	30.6	22.9	12.3	23.0	21.8	21.1	65.5	80.6	39.7	28.5	56.2	54.3	35.7
	H_2O-6x	20.9	23.7	32.4	29.7	21.6	12.7	21.5	21.5	19.7	65.5	81.1	39.8	27.5	53.0	53.3	34.9
	H_2O-8x	19.9	22.3	32.7	29.3	21.3	12.3	20.4	21.0	18.6	65.5	80.6	38.9	28.0	50.2	52.6	34.2
	LLMLingua-2x	15.9	27.6	36.1	28.3	25.4	13.0	28.2	22.4	25.7	6.0	65.7	34.8	19.5	16.2	48.9	27.6
	LLMLingua-4x	14.3	26.3	30.8	27.2	24.0	11.2	25.2	22.1	23.5	1.0	61.9	32.0	5.5	16.0	47.7	24.6
	LLMLingua-6x	14.7	25.6	27.6	24.3	24.7	11.7	23.8	21.6	22.3	0.0	64.4	32.9	5.0	15.3	48.5	24.2
	LLMLingua-8x	14.7	24.7	25.1	23.8	23.5	11.1	23.0	21.4	21.3	0.5	66.5	31.6	6.5	16.7	48.4	23.9

Table 9: Linear-time sequence models and mixed architecture across all datasets in LongBench

LLM	Dataset / Method	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	13.0	6.2	9.1	3.6	17.7	16.3	23.1	50.0	54.0	12.8	1.2	50.5	44.5	20.7
	Mamba-Chat-2.8B	0.4	4.7	0.9	4.1	8.0	0.0	0.7	2.5	1.2	25.5	9.0	0.1	0.0	23.6	17.8	6.6
RWKV	RWKV-5-World-7B	1.3	5.3	6.3	2.4	1.5	0.5	19.2	12.2	18.0	60.5	77.1	41.5	4.0	48.0	40.5	22.6
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	57.2	50.4	26.2
	R-Gemma-9B-it	15.4	25.8	32.3	25.4	27.3	13.0	24.6	18.1	23.0	60.5	70.2	32.4	9.0	63.7	57.6	33.2