RETHINKING KEY-VALUE CACHE COMPRESSION TECHNIQUES FOR LARGE LANGUAGE MODEL SERVING

Wei Gao [*1 2 3] **Xinyu Zhou** [*1] **Peng Sun** [3 4] **Tianwei Zhang** [1] **Yonggang Wen** [1]

ABSTRACT

Key-Value cache (`KV cache`) compression has emerged as a promising technique to optimize Large Language Model (LLM) serving. It primarily decreases the memory consumption of `KV cache` to reduce the computation cost. Despite the development of many compression algorithms, their applications in production environments are still not prevalent. In this paper, we revisit mainstream `KV cache` compression solutions from a practical perspective. Our contributions are three-fold. First, we comprehensively review existing algorithmic designs and benchmark studies for `KV cache` compression and identify missing pieces in their performance measurement, which could hinder their adoption in practice. Second, we empirically evaluate representative `KV cache` compression methods to uncover two key issues that affect the computational efficiency: (1) while compressing `KV cache` can reduce memory consumption, current implementations (e.g., FlashAttention, PagedAttention) do not optimize for production-level LLM serving, resulting in suboptimal throughput performance; (2) compressing `KV cache` may lead to longer outputs, resulting in increased end-to-end latency. We further investigate the accuracy performance of individual samples rather than the overall performance, revealing the intrinsic limitations in `KV cache` compression when handling specific LLM tasks. Third, we provide tools to shed light on future `KV cache` compression studies and facilitate their practical deployment in production. They are open-sourced in https://github.com/LLMkvsys/rethink-kv-compression.

1 INTRODUCTION

The groundbreaking success of Large Language Models (LLMs), e.g., ChatGPT (OpenAI, 2023), Gemini (Hassabis & the Gemini Team, 2023), and Cluade (Anthropic, 2024), is reshaping the global technological landscape. The exponential growth in LLM service requests is creating an unprecedented demand for inference optimization algorithms, which can reduce exorbitant hardware costs and attain superior efficiency. In particular, many research efforts (Sheng et al., 2023; Zhang et al., 2024f) pinpoint that the Key-Value cache (`KV cache`) acts as a major performance bottleneck in LLM serving. For instance, to serve a LLaMA3-70B model with FP16 format, a batch size of 512, and a prompt length of 2048, it requires 130GB of storage space for model weights and an additional 512 GB for the `KV cache`. This high resource requirement underscores the urgent need to mitigate the `KV cache` memory overhead.

There are two prominent strategies to compress `KV cache` and alleviate its exorbitant memory consumption. The first is quantization-based methods (Liu et al., 2024e; Hooper et al., 2024; Kang et al., 2024), which convert the `KV cache` into low-precision representations to reduce the GPU memory usage, albeit with the potential of affecting the accuracy. The effectiveness of these methods depends on the design of representations, which need to strike a balance between memory reduction and maintaining model accuracy. The second is sparsity-based methods (Xiao et al., 2023; Liu et al., 2024d; Zhang et al., 2024f; Li et al., 2024b), which either move less critical portions of the `KV cache` from fast to slow memory or remove them directly. Their success hinges on accurately assessing the importance of the `KV cache` to ensure that the removal of selected entries does not compromise model performance.

Although researchers have demonstrated the effectiveness of `KV cache` compression algorithms, whether they can be deployed in the production environment is still unknown. To bridge this gap, this paper provides an in-depth investigation of existing compression algorithms. We make the following contributions. First, we provide a comprehensive survey of various `KV cache` compression algorithms (Table 1), and summarize relevant benchmark studies (Table 2). Based on the literature review, we identify three key dimensions that have been overlooked in current research evaluation on

[*]Equal contribution [1]Nanyang Technological University [2]S-Lab, Nanyang Technological University [3]Shanghai AI Laboratory [4]SenseTime. Correspondence to: Wei Gao <gaow0007@e.ntu.edu.sg>.

Proceedings of the 8^{th} MLSys Conference, Santa Clara, CA, USA, 2025.

LLM KV cache compression but are crucial for practical deployment. Beyond the model accuracy and GPU memory reduction, assessing the **throughput**, **length distribution**, and **negative samples**[1] can heavily affect the adoption of these compression algorithms in real-world production environments.

Second, we conduct a full-around experimental analysis on representative KV cache compression algorithms, focusing on our identified three dimensions. This brings several novel and interesting observations. (1) KV cache compression algorithms can bring throughput improvement in the decoding stage. However, our evaluation presents poor performance under certain batch sizes and prompt lengths. Besides, when we integrate KV cache compression algorithms with FlashAttention (Dao et al., 2022; Dao, 2024) and PagedAttention (Kwon et al., 2023a), the performance gains from KV cache compression diminish further. (2) The evaluation practice for the computational efficiency of KV cache compression is length controlled. By measuring the length distribution shift induced by KV cache compression, we observe that the lengthier responses produced by these methods can outweigh the throughput speedup benefits, ultimately leading to extended end-to-end latency. (3) We analyze the negative samples in the performance evaluation of KV cache compression methods. We observe their existence across many compression methods and uncover the fragility of KV cache compression towards specific task types.

Third, driven by the limitations of existing KV cache compression algorithms for practical deployment, we provide a set of tools to facilitate their adoption. This includes (1) a throughput analysis tool to decide under which ranges of batch sizes and prompt lengths KV cache compression can present advantageous performance; (2) a length predictor to inform the serving system of requests that could produce longer responses when applying KV cache compression algorithms; (3) a benchmark dataset consisting of our discovered failure cases to help researchers more fairly and accurately assess their compression solutions.

2 Background and Motivation

2.1 Large Language Model (LLM)

A conventional LLM inference process is composed of the following two stages:

Prefill Stage. The prompts are employed to produce KV cache for each transformer layer. Formally, given the input tensor $X \in \mathcal{R}^{b\times l\times d}$, of batch size b, input prompt length l, and hidden dimension d, The key and value tensors are calculated as follows:

$$\begin{aligned} X_K &= X \cdot W_K, \\ X_V &= X \cdot W_V, \end{aligned} \tag{1}$$

where $W_K, W_V \in \mathcal{R}^{d\times d}$ are the weights of the key and value layers, respectively. X_K and X_V are typically cached in the memory, forming KV cache to avoid the re-computation cost in the decoding stage.

Decoding Stage. The model leverages and updates the KV cache dynamically to decode and generate tokens sequentially. We denote the current input token embedding as $x \in \mathcal{R}^{b\times 1\times d}$. The key, value, and query layer outputs are calculated as $x_K = x \cdot W_K, x_V = x \cdot W_V, x_Q = x \cdot W_Q$. Subsequently, we can update the KV cache as follows:

$$\begin{aligned} X_K &\leftarrow [X_K, x_K], \\ X_V &\leftarrow [X_V, x_V], \end{aligned} \tag{2}$$

where $[\cdot]$ denotes the concatenation operation along with the token dimension.

2.2 LLM Serving Acceleration

LLM practitioners commonly adopt two methods to accelerate the LLM serving in production deployment.

FlashAttention (Dao et al., 2022). This is an IO-aware technique designed to reduce the memory overhead in the attention operation. It employs tiling and re-computation strategies alongside an online softmax operation, facilitating tile-based matrix multiplication. The tile size is carefully determined to fit submatrices within the on-chip memory, thereby minimizing the need for data loading to the memory.

PagedAttention. As introduced in vLLM (Kwon et al., 2023b), this technique emulates the virtual memory and paging mechanisms in operating systems for managing the KV cache. It dynamically allocates small GPU memory blocks for the KV cache on demand, rather than pre-allocating memory to the maximum length. This effectively reduces the GPU memory overhead and fragmentation. Usually, PagedAttention is implemented with FlashAttention together to yield the attention output.

2.3 KV Cache Compression

Past works have introduced different techniques to compress the KV cache for LLM serving optimization. They can be roughly classified into the following two categories.

Quantization. These approaches reduce the size of KV cache by using low bits (e.g., INT8) to represent the original full precision (e.g., FP16) without degrading the model accuracy. The quantization and de-quantization processes

[1] This refers to the samples that are benign in the original LLMs, but turn to malign when KV cache compression is applied.

can be illustrated as follows:

$$\begin{aligned} \text{quantize:} \quad & X_{\text{quant}} = \lfloor \frac{X - \ell}{\Delta} \rceil, \quad \Delta = \frac{u - \ell}{2^b - 1}, \\ \text{de-quantize:} \quad & \hat{X} = X_{\text{quant}} \cdot \Delta + \ell, \end{aligned} \tag{3}$$

where X is the original KV cache, X_{quant} is the quantized result from X, and $\hat{X}$ is the de-quantized result from X_{quant}. These quantization-based methods aim to minimize the difference between X and $\hat{X}$. As the quantized KV cache does not participate in the prefill stage, quantization typically happens in the decoding stage to reduce the memory consumption of token generation.

Sparsity. These approaches are inspired by the sparsity of attention scores (Xiao et al., 2023; Liu et al., 2024d; Zhang et al., 2024f; Li et al., 2024b; Yang et al., 2024a; Zhang et al., 2024d). They dramatically reduce the memory footprint of the KV cache by evicting the KV pairs of less important tokens while retaining the ones of more important tokens to preserve the model's accuracy. The key to these solutions is to determine when and how many tokens should be evicted for effective compression.

2.4 Concerns in LLM Production Deployment

Despite the memory reduction benefits, the main concerns that prevent LLM practitioners from deploying KV cache compression methods in production environments are accuracy and computational efficiency.

For accuracy, many compression studies emphasize minor overall accuracy loss while concealing the specific accuracy performance defects of KV cache compression methods. Furthermore, many techniques possess abundant configurations to balance the accuracy performance and memory compression ratio, increasing the burden of tuning the complex configuration parameters.

For computational efficiency, LLM practitioners focus on the key metrics of Time-To-First-Token (TTFT) and Time-Between-Output-Token (TBOT). However, they are typically measured using the naive transformers library (TRL) (Wolf et al., 2020), which falls short of the performance expected in production environments. Besides, the speedup benefits from KV cache compression are not always guaranteed. In particular, during the decoding stage, compressing the KV cache primarily influences the attention operation. The time required for the attention operation to generate one token is comprised of two steps: (1) loading data into on-chip memory and (2) performing the computation. The reduction of memory consumption brought by KV cache compression can lead to decreased time for the first step. However, for quantization-based methods, the operation described in Eqn. 3 incurs additional overhead for the second step. For sparsity-based methods, although the reduced KV cache size can lower the time cost spent on the second step, an extra computation step is required for the KV cache eviction, with additional time overhead.

3 A Comprehensive Survey

3.1 KV Cache Compression Algorithms

Table 1 summarizes existing quantization- and sparsity-based KV cache compression algorithms. We focus on the design components of these algorithms that could adversely affect computational efficiency.

3.1.1 Quantization

We first review some relevant KV cache quantization algorithms, focusing on the quantization granularity and quantization error. First, a few studies observe that the sensitivity to quantization operations varies with the granularity of KV cache. ZipCache (He et al., 2024) and WKVQuant (Yue et al., 2024) adopt channel-separable token-wise quantization for KV cache. KVQuant (Hooper et al., 2024), and KIVI (Liu et al., 2024e) utilize per-channel quantization for the key tensor and per-token quantization for the value tensor. QJL (Zandieh et al., 2024) introduces the quantized JL transform to key tensor and per-token quantization for value tensor to reduce the memory consumption of the quantized tensor. MiKV (Yang et al., 2024b), QAQ (Dong et al., 2024b), and SKVQ (Duanmu et al., 2024) allow varying bits to represent KV cache to attain accuracy and memory reduction balance. Coupled Quantization (Zhang et al., 2024b) integrates multiple channels and jointly quantizes them. Additionally, it leverages Fisher information to prioritize important tokens when quantizing. From an accuracy standpoint, researchers are advancing dedicated quantization operations toward finer KV cache granularity. **While finer quantization granularity may preserve the accuracy performance, it introduces irregular computational patterns, thereby limiting the effective utilization of GPU resources.** Consequently, these approaches may not lead to improved computational performance.

Second, some research efforts aim to rectify the quantization error to sustain the LLM response quality. For example, Gear (Kang et al., 2024) approximates the quantization error with a low-rank matrix. Quantization outliers, which manifest as quantization errors with extreme values, can greatly impair model performance. To counter this, IntactKV (Liu et al., 2024c) retains the full precision of the outlier to maintain the model accuracy, while Gear employs a sparse matrix to mitigate the error caused by the outliers. Additionally, QuaRot (Ashkboos et al., 2024) innovatively transforms each weight matrix with Hadamard orthogonal matrices to eliminate the quantization outlier without changing the LLM output. Addressing the quantization error remains a promising area to improve the accuracy of quantization-

Table 1. Summary of existing KV cache compression algorithms.

Date (YY.MM)	Algorithm	Quant / Sparse	Algorithm Features	Model	Heavy Eval	Mem	Prf Thr	Dec Thr	Frw
24.02	KVQuant (Hooper et al., 2024)	Q	Per-channel key quantization	L	65B / 1 / 32k	8.0 ×	-	-	T
24.02	WKVQuant (Yue et al., 2024)	Q	Loss design for quant parameter optimization	L	13B / 16 / 18k	4.0 ×	-	-	T
24.02	KIVI (Liu et al., 2024e)	Q	Per-channel key quantization	L, M, F	13B / 380 / 18k	2.6 ×	2.3 ×	3.4 ×	T
24.02	MiKV (Yang et al., 2024b)	Q	Mixed-precision quantization	L, M	70B / 8 / 4k	5.0 ×	-	-	T
24.03	IntactKV (Liu et al., 2024c)	Q	Keep full-precision caches for outlier tokens	L, M	70B / 1 / -	4.0 ×	-	-	T
24.03	QAQ (Dong et al., 2024b)	Q	Quality-adaptive quantization	L	13B / 1 / -	10.0 ×	-	-	T
24.03	GEAR (Kang et al., 2024)	Q	Approximate the quant error with low-rank matrix	L, M	13B / 18 / 7k	3.8 ×	-	5.0 ×	T
24.03	QuaRot (Ashkboos et al., 2024)	Q	Eliminate KV outliers with Hardmard matrix	L	70B / 64 / 2k	3.7 ×	2.1 ×		T
24.05	SKVQ (Duanmu et al., 2024)	Q	Clipped dynamic quant with channel reorder	L, M	13B / 128 / 200k	7.9 ×		7.0 ×	T
24.05	ZipCache (He et al., 2024)	Q	Channel-separable tokenwise quantization	L, M	13B / 8 / 4k	4.9 ×	1.6 ×	2.3 ×	T
24.07	QJL (Zandieh et al., 2024)	Q	Elimiate quant constants storage overheads with JL transform	L	8B / 1 / 18k	5.2 ×	-	-	T
24.07	Palu (Chang et al., 2024)	Q	KV cache compression with low-rank projection	L, M	13B / 1 / 64k	11.4 ×	-	1.6 ×	T
24.08	ZDC (Zhang & Shen, 2024)	Q	Eliminate compression overhead	O, L	175B / 1 / 20k	10.0 ×	-	2.8 ×	T / D / V
23.08	Scissorhands (Liu et al., 2024d)	S	Window-based eviction with a counter-based token score	O	175B / 128 / 2k	5.0 ×	-	-	T
23.12	StreamingLLM (Xiao et al., 2023)	S	Retain KV cache of initial tokens	L, F, M	70B / 1 / 18k	5.0 ×	-	-	T
23.12	H2O (Zhang et al., 2024f)	S	Accumulate attention scores as token score	L, O, G	66B / 64 / 7k	5.0 ×	-	29.0 ×	T / D / F
24.01	FastGen (Ge et al., 2023)	S	Head-adaptive eviction policy	L	65B / 16 / 4k	1.6 ×	-	1.2 ×	T / D / F
24.02	LESS (Dong et al., 2024a)	S	Merge to-be-evicted caches into low-rank matrix	L, F	13B / 64 / 5k	50.0 ×	-	1.7 ×	T
24.02	ROCO (Ren & Zhu, 2024)	S	Standard deviation of attention score as token score	L, W	7B / 1 / -	3.3 ×	-	-	T
24.04	Keyformer (Adnan et al., 2024)	S	Add gumbel-based regularization in token score	G	7B / 2 / 4k	2.0 ×	-	2.4 ×	T
24.04	SqueezeAttention (Wang & Gan, 2024)	S	Reallocate KV cache budget across layers	L, M, F, G, O	70B / 224 / 18k	3.3 ×	-	2.2 ×	T
24.04	SnapKV (Li et al., 2024b)	S	Select clustered important KV cache across heads	L, M	35B / 8 / 26k	8.2 ×	-	3.6 ×	T
24.04	CORM (Dai et al., 2024)	S	Budget-unrestricted KV cache eviction	L	7B / 1 / 18k	3.3 ×	-	-	T
24.05	CaM (Zhang et al., 2024c)	S	Merge to-be-evicted caches into recent KV cache	L, O, G	13B / 1 / -	3.3 ×	-	-	T
24.05	PyramidInfer (Yang et al., 2024a)	S	Drop KV cache during KV cache computation process	L	70B / 88 / 2k	2.1 ×	-	2.2 ×	T / D
24.05	MiniCache (Liu et al., 2024a)	S	Multiple layers sharing the same retained KV cache	L, M	70B / 300 / 18k	1.7 ×	-	5.0 ×	T
24.05	InfLLM (Xiao et al., 2024a)	S	Store evicted tokens as context memory for furthur lookups	L, M	8B / 1 / 100k	2.9 ×	-	1.5 ×	T
24.05	Q-Hitter (Zhang et al., 2024e)	Q + S	Keep quantization-friendly and important tokens	L, O	30B / 1 / 4M	20 ×	-	33.0 ×	T
24.06	Quest (Tang et al., 2024b)	S	Query-aware cache eviction policy	L	7B / 1 / 64k	8.0 ×	-	2.2 ×	F
24.06	PyramidKV (Zhang et al., 2024d)	S	Adjust KV cache budget across layers	L, M	8B / 1 / 18k	8.3 ×	-	-	T
24.06	SampleAttention (Zhu et al., 2024)	S	Adaptive structured sparse attention	C, I	6B / 1 / 200k	12.5×	2.2 ×	-	T
24.07	TOVA (Oren et al., 2024)	S	Enable recent KV cache evictable	L	7B / 139 / 70k	-	-	4.8 ×	T
24.07	LazyLLM (Fu et al., 2024)	S	Revive previously evicted KV cache	L	7B / 1 / 18k	-	2.3 ×	-	T
24.07	Ada-KV (Feng et al., 2024)	S	Allocate KV cache budget across different heads	L, M	7B / 1 / 18k	3.3 ×	-	-	T
24.07	RazorAttention (Tang et al., 2024a)	S	Disable KV cache eviction for retrieval heads	L, Q, B	72B / 1 / 18k	3.3 ×	-	-	T
24.07	ThinK (Xu et al., 2024)	S	Evict KV cache in channel dimension	L, M	8B / 1 / 18k	1.25 ×	-	-	T
24.08	NACL (Chen et al., 2024)	S	General KV cache eviction framework	L	7B / 4 / 32k	5.0 ×	-	-	T
24.08	DoubleSparse (Yang et al., 2024c)	S	Prefetch tokens with token and channel sparsity	L, M	70B / 32 / 256k	16 ×	-	16.3 ×	T
24.09	GemFilter (Shi et al., 2024)	S	Use early layes of LLM to filter and compress tokens	L, M	12B / 1 / 120k	1.43 ×	-	2.4 ×	T
24.09	RetrievalAttention (Liu et al., 2024b)	S	Leverage vector search for dynamic sparse attention	L	8B / 1 / 1M	-	-	4.9 ×	T
24.10	DuoAttention (Xiao et al., 2024b)	S	Identify streaming heads to accelerate attention	L, M	8B / 1 / 3.3M	2.55 ×	1.73 ×	2.18 ×	F

Notations: In column **Model**, L, M, F, O, C, and I represent Llama, Mistral, Falcon, OPT, ChatGLM, and the InternLM LLM family. In column **Heavy Eval**, we list the heaviest evaluation setting in a format of model size/ batch size/ prompt length. We list the maximum memory reduction, prefill throughput speedup, and decoding throughput speedup in column **Mem**, **Prf Thr**, and **Dec Thr**. For evaluation frameworks shown in column **Frw**, T, D, F, and V represent the transformer library (Wolf et al., 2020), DeepSpeed (Aminabadi et al., 2022), FlashInfer (Flashinfer, 2024), and vLLM (Kwon et al., 2023b).

based methods. However, error correction necessitates another step to compute the quantization error. **Mitigating the quantization error not only compromises the memory efficiency but also incurs additional computational costs during inference, which can offset the computational benefits from memory reduction.**

Note that most quantization-based algorithms keep a window of recent historical KV cache as full precision for accuracy considerations, which could burden the compatibility with PagedAttention. Specifically, PagedAttention maintains a number of tensors with a fixed page size and tensor type. The window-based quantization demands two types of paged tensors to manage full-precision and quantized KV cache, respectively. This introduces unstructured computation patterns when computing the attention output. **The window-based design choice in quantization-based methods increases the deployment complexity, potentially negating the computational efficiency gains.**

3.1.2 Sparsity

We further review sparsity-based KV cache compression algorithms from two perspectives. First, we investigate the granularity of KV cache compression, encompassing token-level, layer-level, head-level, and channel-level. Early works including Scissorhands (Liu et al., 2024d), StreamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2024f), ROCO (Ren & Zhu, 2024), Keyformer (Adnan et al., 2024) propose discarding the KV cache of unimportant tokens. Layer-level KV cache eviction methods, e.g., SqueezeAttention (Wang & Gan, 2024), PyramidKV (Zhang et al., 2024d), and MiniCache (Liu et al., 2024a), enable the selective removal of KV cache entries corresponding to different token positions across layers. FastGen (Ge et al., 2023), SnapKV (Li et al., 2024b), CORM (Dai et al., 2024), Ada-KV (Feng et al., 2024), RazorAttention (Tang et al., 2024a), and NACL (Chen et al., 2024) evict varying token positions across different heads. Notably, ThinK (Xu et al., 2024) deviates from these approaches by targeting partial channel dimensions, thereby achieving a consistent reduction in KV cache size, irrespective of the sequence length. PQCache (Zhang et al., 2024a) utilizes Product Quantization (PQ) to manage the KV cache. It employs Maximum Inner-Product Search (MIPS) to identify relevant tokens for attention computations during the decoding stage. **Similar to quantization-based techniques, pursuing finer granularity in sparsity-based methods can yield improved accuracy, at the cost of high GPU utilization owing to the resulting irregular computational patterns.**

Second, we summarize the eviction policies, focusing on the importance metric, eviction scope, and budget allocation. (1) The **importance metric** is to determine the relative order by which caches are evicted. Many attention score

variants (Liu et al., 2024d; Ge et al., 2023; Ren & Zhu, 2024; Adnan et al., 2024; Tang et al., 2024b) are used as the importance metric. The accumulated attention score is particularly popular due to its robust accuracy performance over diverse LLM tasks. (2) The **eviction scope** defines the tokens eligible for eviction. A prevalent approach is to employ a local window to reduce the eviction overhead. Additionally, constraints are often considered to ensure that specific tokens (Xiao et al., 2023; Oren et al., 2024), heads (Tang et al., 2024a), or layers (Tang et al., 2024b) are excluded from the eviction scope, safeguarding model quality. (3) The **budget allocation** dictates how the available GPU memory budget is allocated. A straightforward solution is to set a fixed memory budget; however, not all layers and heads are equally important. Therefore, some methods dynamically allocate the memory budget across layers (Wang & Gan, 2024; Yang et al., 2024a; Zhang et al., 2024d) and heads (Feng et al., 2024). Moreover, CORM (Dai et al., 2024) introduces a budget-unrestricted `KV cache` eviction policy. The eviction policy of sparsity-based methods is more flexible and complex than the quantization policy as described in Eqn. 3.

The pursuit of fine-grained token eviction and intricate eviction policies intensifies the challenges of gaining computation efficiency from sparsity-based methods. These concerns remain consistent across various LLM serving frameworks. Moreover, our analysis reveals that the algorithmic designs of sparsity-based methods are not compatible with FlashAttention and PagedAttention. FlashAttention gets rid of the multi-pass attention operation and exploits softmax and tiling to attain a one-pass operation, thus reducing the number of passes to load data from global memory to on-chip memory. However, the importance metric depends upon the attention scores, which are not saved in the FlashAttention process. As a result, calculating the importance metric necessitates two additional passes to load the data and one more step to compute the attention scores. PagedAttention usually assumes that the length of `KV cache` for a request monotonically increases. Sparsity-based methods execute token eviction at a fixed interval. Thus, the length of the remaining `KV cache` fluctuates over time, exacerbating the complexity of `KV cache` management with PagedAttention. **The integration of sparsity-based methods with FlashAttention and PagedAttention increases the implementation complexity and compromises potential computational efficiency advantages, underscoring the necessity for targeted systematic optimizations.**

3.1.3 Evaluation Settings

Table 1 collects the evaluation settings of various `KV cache` compression algorithms. First, many research works spare more empirical analysis on accuracy than computational efficiency. The research trends that incorporate undesirable design elements in `KV cache` compression may preserve the accuracy but fail to facilitate computational gains because these design elements are not sufficient to exploit GPU parallelism. Notably, many studies only report the throughput for assessing the computational efficiency of the TRL framework, yet the design of compression algorithms possesses features that may not align with established optimizations for LLM serving. As a result, the potential for performance speedup becomes obscured when deploying compression algorithms in production environments.

Second, around half of the quantization-based algorithms are evaluated on models with a maximum size of 13 billion parameters and a maximum sequence length of 20 thousand. Statistically, more sparsity-based works evaluate larger model sizes (70B) and longer prompt lengths (200K) than quantization-based ones. This necessitates tensor parallelism to support such extreme scenarios on multiple GPUs.

> **Missing Piece 1**: With less focus on computational efficiency, only a few compression studies measure the throughput performance using the TRL framework, often neglecting serving techniques, including FlashAttention and PagedAttention. Additionally, there is a lack of assessment regarding the throughput on multiple GPUs, which is crucial for supporting large models and long sequences through tensor parallelism.

Third, the throughput performance of different compression algorithms is typically evaluated with a fixed response length. The response length is a crucial factor that impacts the end-to-end latency of LLM serving requests. The response length is known when a termination token (e.g., `EOS`) is present. Thus, measuring the computational efficiency with a fixed response length is not an appropriate approach. In a realistic macro-benchmark, it is essential to account for the variations in the length distribution difference and assess the end-to-end latency performance of `KV cache` compression techniques for a fair comparison.

> **Missing Piece 2**: The end-to-end latency variations from compression algorithms hinge not only on the throughput but also on the response length. However, the effect of compression algorithms on the response length has been largely neglected in existing compression studies.

3.2 `KV Cache` Compression Benchmark Studies

The surge of `KV cache` compression algorithms drives several benchmark studies. We review them in Table 2 and dissect them from tasks, models, and metrics. We identify several insights and pinpoint a gap in the accuracy performance demonstration within current studies.

First, for evaluation metrics, only LLM-QBench (Gong et al., 2024) measures the prefill and decoding throughput.

Table 2. Summary of KV cache compression benchmarks. W/A/KV denote weights, activations, and KV cache, respectively.

Benchmarks	Tasks	Models	Metrics	Eval Methods	Benchmark Features
QLLM-Eval (Li et al., 2024a)	Basic NLP Tasks Emergent Ability Trustworthiness Dialogue Long-context Tasks	OPT LLaMA2 Falcon Mistral	Acc	W/A/KV Quant	Evaluation of quant impacts for efficient deployment
LLM-QBench (Gong et al., 2024)	WikiText2, C4 Exam & Coding	LLaMA2 ChatGLM CodeLLaMA WizardMath	Acc Throughput	W/A/KV Quant	A plug-and-play toolkit to explore quant impacts
LongCTX-Bench (Yuan et al., 2024)	Long-context Tasks	Mistral LLaMA	Acc	KV Quant KV Sparse	KV compression evaluations in a long-context environment
Shi et al. (2024) (Luohe et al., 2024)	Basic NLP Tasks	LLaMA	Acc	KV Quant KV Sparse	A systematic review on KV optimizations without empirical results

These benchmark studies generally suggest that compression algorithms result in only minor overall performance degradation. However, the aggregated numerical value may obscure the biases introduced by KV cache compression toward specific tasks and sample types. **Existing studies lack an in-depth analysis of how KV cache compression affects the response quality of individual examples.**

Second, for evaluation models, accuracy performance analysis has been primarily conducted using LLM families such as LLaMA (Touvron et al., 2023) and Mistral (Jiang et al., 2023). Their guidance on KV cache compression techniques is derived from empirical studies across LLM families. For example, LLM-QBench (Li et al., 2024a) recommends adjusting the quantization bit-widths based on the specific LLM family. **This underscores the importance of investigating accuracy performance within the LLaMA and Mistral families to gain comprehensive insights.**

Third, for evaluation tasks, a benchmark demands a comprehensive understanding of the LLMs' capabilities in a multidimensional manner. Existing studies (Li et al., 2024a; Gong et al., 2024; Luohe et al., 2024) assess a wide range of language tasks, including language understanding, modeling and reasoning, emergent abilities, dialogue, and long-context tasks. According to these analyses, long-context tasks exhibit lower tolerance to KV cache compression. **A detailed response quality analysis to individual samples in long-context tasks is imperative to understand the limitations of KV cache compression.**

Missing Piece 3: Long-context tasks are challenging for KV cache compression algorithms to maintain the accuracy. Many works report the overall performance, but they often overlook the analysis of response quality for individual samples.

4 EVALUATION

In this section, we evaluate KV cache compression, with the focus on three missing aspects: *throughput analysis*, *response length distribution*, and *negative sample analysis*.

4.1 Evaluation Recipes

We single out representative models, datasets, and algorithms based on the above literature analysis. The majority of the experiments are conducted on A6000 GPUs with LLM serving frameworks, including TRL, FlashAttention (Dao, 2024), and LMDeploy (Contributors, 2023). We have also extended the scope to H800 GPU in Figure 2. We select LMDeploy for its efficient quantization kernels and fast development of KV cache compression algorithms. Appendix B offers additional explanations and comprehensive details about the evaluation setup, further validating the rationale behind the selection of LMDeploy.

LLMs. Inspired by previous benchmark studies and implementation complexity, our evaluation only chooses LLaMA and Mistral families. We integrate corresponding compression algorithms and LLMs into TRL and LMDeploy.

Datasets. We choose ShareGPT (Anon, 2024) to conduct the throughput analysis. This dataset is commonly used for LLM-serving benchmarks in real-world applications. We choose LongBench (Bai et al., 2023) to perform negative sample analysis. This is a task for long-context understanding that covers critical long-text application scenarios.

Metrics. For computational efficiency, we concentrate on the throughput and end-to-end latency. In particular, we report pre-fill and decoding throughput using synthesized examples and measure the end-to-end latency using samples from ShareGPT. For negative sample analysis, LongBench provides task-specific metrics to evaluate the accuracy of individual samples.

Compression Algorithms. We choose four compression algorithms for evaluation. For quantization-based ones, KIVI (Liu et al., 2024e) is a mainstream solution for quantizing key and value tensors. GEAR (Kang et al., 2024) is a popular quantization error mitigation algorithm. For sparsity-based ones, StreamingLLM (Xiao et al., 2023) only keeps first and recent tokens without complex attention score computation and presents a structured computation pattern. H2O (Zhang et al., 2024f) dynamically evicts KV cache with complex attention score computations.

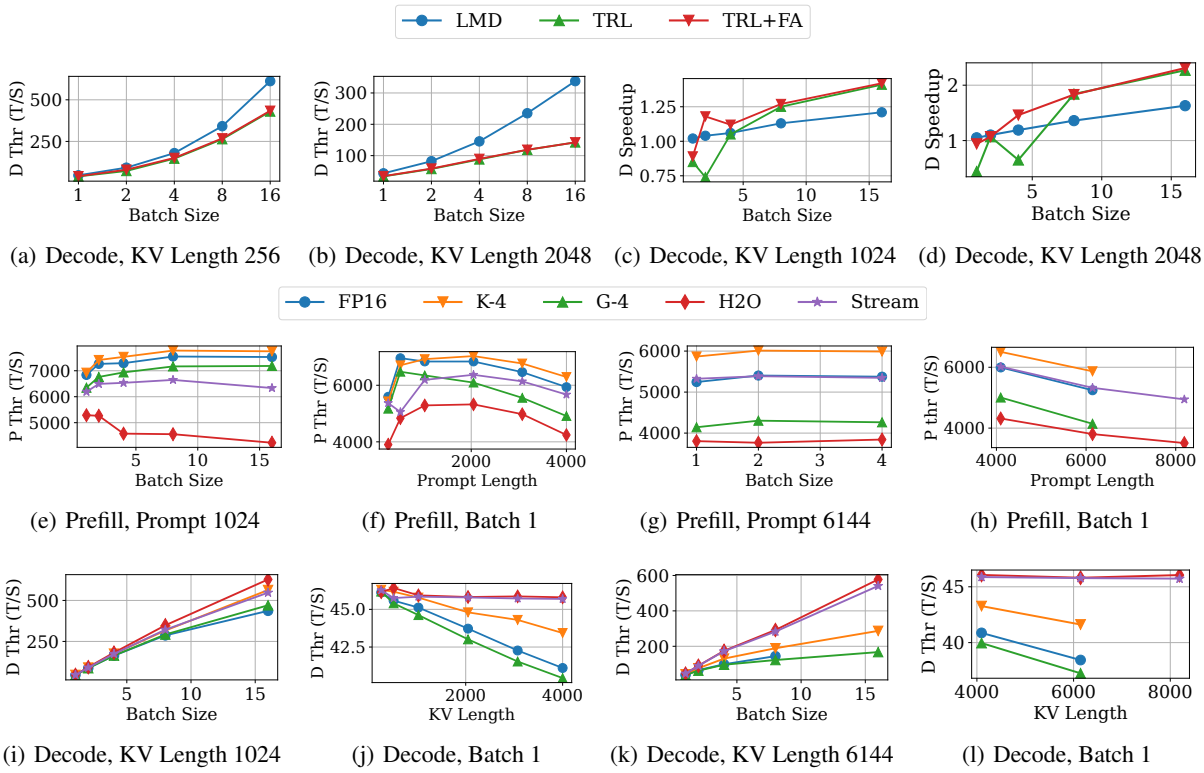


Figure 1. Throughput analysis of LLaMA-7B: (a-b) The FP16 decoding throughput on TRL (with and without FlashAttention) and LMDeploy (LMD). (c-d) The speedup of the StreamingLLM algorithm on TRL and LMD. (e-h) The prefill throughput for various sizes of inputs. (i-l) The decoding throughput for various sizes of inputs.

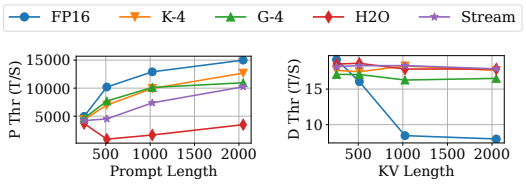


Figure 2. Throughput analysis of LLaMA-70B on H800 GPUs.

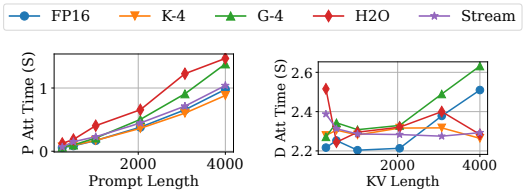
4.2 Throughput Analysis

We first utilize synthesized samples and LLaMA-7B to analyze the prefill and decoding throughput of LLM serving acceleration techniques. We measure the decoding throughput of full-precision baseline on TRL, TRL+PagedAttention, and LMDeploy, which possess the functionality of PagedAttention and FlashAttention. As shown in Figure 1 (a-b), PagedAttention and FlashAttention can improve the decoding throughput. Moreover, Figure 1 (c-d) shows the relative speedup of decoding throughput between the FP16 baseline and StreamingLLM across batch sizes by fixing `KV` lengths. When the batch size exceeds 4 and the sequence length reaches 1024, the relative speedup on TRL does not show a significant advantage when measured against PagedAttention and FlashAttention.

Observation 1: The computational efficiency results on TRL are unreliable. An appropriate way is to measure the throughput performance on established LLM serving frameworks with prominent LLM serving techniques, including PagedAttention and FlashAttention.

Second, we compare the prefill throughput of different compression algorithms. In particular, we collect the prefill throughput performance across different batch sizes and prompt lengths in Figure 1 (e-h). KIVI and StreamingLLM perform close to and even better than the FP16 baseline. However, GEAR and H2O consistently lower the prefill throughput, with the gap widening as the prompt length increases. Qualitatively, GEAR introduces extra steps to offset the quantization error, and H2O requires a multi-pass attention operation to compute the attention score as a result of high memory access overhead. The execution time of the attention layer in GEAR and H2O in Figure 3(a) further demonstrates that the additional `KV cache` compression overhead should not be disregarded in the prefill stage.

Third, we measure the decoding throughput for different compression algorithms, shown in Figure 1 (i-l). The throughput difference is insignificant when both batch size and KV length are small among different methods, including the baseline, as illustrated in Figure 1 (i-j). For heavy settings with long KV length and high batch size, sparsity-



(a) Prefill, Batch 1 (b) Decode, Batch 1

Figure 3. The execution time of the attention layer of various compression algorithms measured across different prompt lengths.

based methods can maintain their throughput advantages, whereas the benefits of quantization-based methods tend to diminish. We also observe that quantization-based methods even suffer from out-of-memory issues when the KV length reaches up to 8192 in Figure 1 (l). Moreover, Figure 3(b) depicts the execution time of the attention layer in sparsity-based methods remains more stable during the decoding stage across various KV lengths, as they retain a relatively small `KV cache`. For larger LLMs, we observe a similar phenomenon in decoding throughput, with results provided in Appendix C.

Table 3. Relative speedup brought by different compression algorithms in the prefill and decoding.

	TP/Algo	FP16 (Thr: T/S)	K-4	G-4	H2O	Stream
Prefill	1	6610.24	1.06×	0.86×	0.58×	0.95×
	2	11041.35	1.09×	0.80×	0.58×	0.96×
	4	12938.66	1.03×	0.90×	0.51×	0.92×
Decode	1	129.72	0.98×	1.02×	1.34×	1.34×
	2	194.83	0.88×	0.97×	0.69×	1.01×
	4	195.02	0.9×	0.97×	0.85×	0.97×

Fourth, we report the relative speedup in the prefill and decoding throughput of different compression algorithms compared to the FP16 baseline across various tensor parallelism settings in Table 3. While tensor parallelism can improve throughput, it may also reduce the throughput speedup gained from `KV cache` compression and, in some cases, even negatively impact overall throughput performance. We ascribe this to that increasing tensor parallelism can alleviate memory bandwidth contention on each GPU, thereby weakening the benefits of reduced memory access overhead achieved through `KV cache` compression. Note that Appendix C consists of more throughput analyses across different batch sizes, sequence lengths, LLMs (e.g., Mistral (Jiang et al., 2023) in Figure 8), and algorithms (e.g., SnapKV (Li et al., 2024b) in Figure 9), our conclusions remain consistent.

Observation 2: The `KV cache` compression methods show negative computational efficiency in certain scenarios of batch size, sequence length, and tensor parallelism in the prefill and decoding stage. We recommend applying compression algorithms for serving requests with heavy `KV cache`.

Table 4. Comparison of semantic scores and length increase for different `KV cache` compression algorithms.

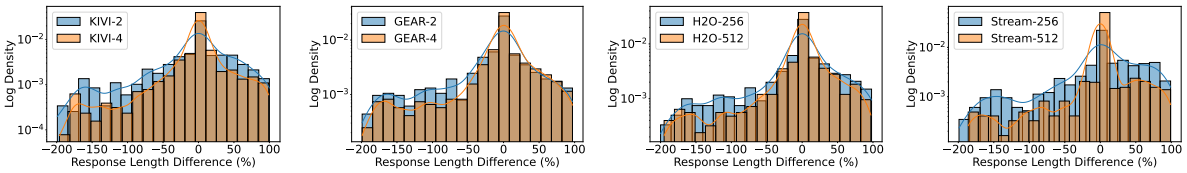
Metric	FP16	KIVI-4	GEAR-4	H2O-512	Stream-512
Semantic Score	49.6	50.7	46.2	46.2	46.3
Length Increase (×)	-	1.69	1.70	1.55	1.76

4.3 Length Distribution Analysis

Lossy compression can elicit LLMs to yield different lengths of responses. Despite potential throughput benefits from `KV cache` compression, the lengthy responses can still prolong the end-to-end latency. We first examine the response length difference induced by compression methods. We define the response length difference as $D = (L^{\text{un}} - L^{\text{cs}})/L^{\text{un}}$, where L^{un} and L^{cs} represent the response length without and with compression methods, respectively. A negative D implies that compression methods lead to longer outputs, while a positive D suggests shorter responses due to compression. We use 1,000 samples from ShareGPT and measure D on LLaMA-3.1-8B-instruct using different `KV cache` compression algorithms. First, compression algorithms tend to increase the response length. We gather samples whose response length either decreases or increases by 50%. Table 5 presents the proportion of samples exhibiting substantial variations in response length. Since the temperature in text generation affects response length, we use temperatures of 0.9 and 1.1 for a fair comparison while fixing the temperature as 1.0 for the baseline and other compression algorithms. We observe that the temperature can increase and decrease the response length in roughly equal measure. Differently, `KV cache` compression leans toward generating lengthy responses. Specifically, more than 20% of the samples show at least a 1.5 × increase in response length. Prior throughput analysis shows that compression methods cannot achieve more than a 1.5× increase in the decoding throughput in many scenarios. This indicates that these samples will suffer from increased end-to-end latency due to their extended response length.

Second, we investigate the impact of compression ratio on the length difference distribution. A higher compression ratio can be achieved with a lower bit in the quantization-based method or a lower length of `KV cache` in sparsity-based methods. We draw the distribution of the difference in response length (bar) and approximate the kernel density estimation of this distribution (line) across varying compression ratios in Figure 4. We observe that with the increase of the compression ratio, the distribution of response difference flattens, and more samples experience lengthy response. The lengthy responses may serve as an implicit way for `KV cache` compression to compensate for accuracy loss.

To uncover whether `KV cache` compression techniques yield verbose output to improve the accuracy, we provide an intuitive experiment to investigate the verbosity of out-



(a) KIVI (b) GEAR (c) H2O (d) StreamingLLM

Figure 4. The log-scaled distribution of response length difference over different compression algorithms and configurations.

Table 5. The ratio (%) of samples experiencing response length variations induced by temperature and `KV cache` compression.

Metric	T=0.9	T=1.1	KIVI	GEAR	H2O	Stream
% of samples D of which $\geq 50\%$	20.8%	21.3%	10.9%	6.8%	9.5%	16.5%
% of samples D of which $\leq -50\%$	27.5%	31.4%	24.5%	27.1%	21.3%	26.4%

puts. We define the verbose output as follows: for the output of an LLM with FP 16 baseline, its output quality and length are defined as Q_{fp16} and L_{fp16}; for the output of compression baseline, its accuracy and length is defined as $Q_{compress}$ and L_{fp16}. We consider the output of compression baseline is verbose when $Q_{compress} \leq Q_{fp16}$ and $L_{compress} \geq L_{fp16}$. We choose 200 requests from ShareGPT, in which KV cache compression techniques yield longer responses than FP 16 baseline when adopting LLaMA-7B. Fortunately, ShareGPT provides real-world conversations between humans and ChatGPT. To evaluate the semantic similarity between the outputs of ChatGPT and LLaMA-7B. We report the average semantic score and relative length increase in Table 4. We find that three KV cache compression approaches produce longer outputs but with relatively minor semantic quality drops. Compared to the output from an LLM with FP 16, the output from an LLM with a KV cache compression approach might be more verbose.

Observation 3: The lossy compression contributes to a large variation of response length distribution. Compression algorithms produce lengthy responses. Furthermore, a high compression ratio exacerbates this issue.

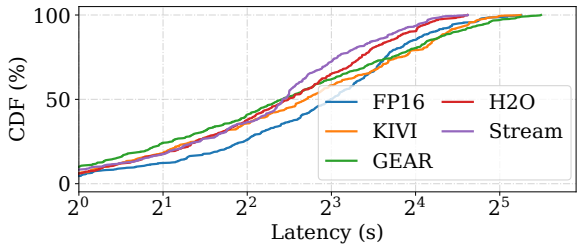


Figure 5. The cumulative distribution function of the end-to-end latency (seconds) of various compression algorithms.

Third, beyond the length distribution analysis, we present the distribution of end-to-end latency to gain a vivid understanding of how length difference impacts computational efficiency. Specifically, we measure the end-to-end latency performance for each sample in the ShareGPT dataset with a fixed batch size of one, as shown in Figure 5. When we combine the throughput and response length, the performance gains of compression methods are not significant. We even observe that GEAR leads to longer end-to-end tail latency. In other words, only measuring the throughput performance with a fixed response length cannot convince LLM practitioners to deploy compression algorithms in production environments. We leave more empirical analysis about length distribution in Appendix D.

Observation 4: Measuring the end-to-end latency uncovers that compression methods still have a long way to go in practice. In addition to the innate algorithm defect of compression that might lower the throughput performance benefits, the lengthy response length is another factor that hinders their adoption.

4.4 Negative Sample Analysis

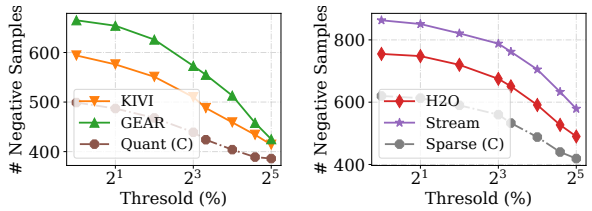
Many studies on `KV cache` compression place emphasis on preserving the quality of LLM responses, supported by empirical evidence from a few benchmark datasets. While most of them report minor/no accuracy performance drops, they often overlook the impact of compression on individual samples. In other words, LLM practitioners lack insightful guidance on when `KV cache` compression fails to achieve satisfactory performance. To bridge such a gap, we explore the negative performance impact on individual samples. We follow the criterion in QLLM-Eval (Li et al., 2024a) to define the *negative sample* as a benign sample where `KV cache` compression leads to the relative accuracy loss exceeding a given threshold[2]. Algorithm 1 describes the process of collecting negative samples given the LLM, dataset, and compression algorithms. The baseline algorithm is the one without using `KV cache` compression.

First, we unveil the fragility of compression algorithms using negative samples. We use LongBench and LLaMA-3.1-8B-instruct to conduct negative sample analysis. The experimental details can be found in Appendix E. We vary the threshold in Algorithm 1 to collect negative samples shown in Figure 6. The minor accuracy loss brought by compression algorithms (e.g., KIVI, GEAR) does not mean

[2]In the evaluation, we select samples with accuracy equal to or above the average value as benign.

that each sample suffers from the minor performance loss. Our pinhole observation indicates a high number of negative samples even with a threshold of 10%, revealing the fragility of compression algorithms. Combining an ensemble of algorithms to construct an algorithm set $\mathcal{A}$ is a feasible approach to reduce the occurrence of negative samples; however, they cannot be totally eliminated.

Observation 5: `KV cache` compression algorithms naturally possess negative samples, and the accuracy improvement can reduce the occurrence of negative samples, but it is hard to eradicate.



(a) Quantization (b) Sparsity

Figure 6. The threshold (x-axis) versus the number of negative samples (y-axis) for quantization-based (a) and sparsity-based (b) methods. Quant (C) refers to negative samples collected using both KIVI and GEAR together. Sparse (C) refers to negative samples collected using both H2O and StreamingLLM together.

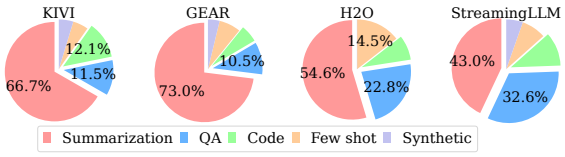


Figure 7. The pie chart details the proportion of negative samples over task types across varying compression algorithms.

Second, we explore the sensitivity of task types to `KV cache` compression using a threshold of 10%. Figure 7 depicts the breakdown of the number of negative samples across various tasks. Both quantization-based and sparsity-based methods exhibit a similar unbalanced fragility across different task types. Particularly, the summarization tasks depend heavily on context information, and any loss of this information can lead LLMs to produce undesirable responses. Similarly, in the question-answering (QA) task, information lost in the early stages can become significant in later stages, thereby amplifying this issue. Fortunately, a recent work, Quest (Tang et al., 2024b), proposes a query-aware approach to address this drawback.

Observation 6: Not all task types are equally treated by `KV cache` compression algorithms. The compression studies show limitations in maintaining accuracy for summarization and QA tasks.

Algorithm 1 Negative Sample Collection.

Input: Dataset $\mathcal{D}$, Threshold θ, LLM $\mathcal{M}$, Baseline Algorithm $\mathcal{A}_\text{b}$, Compression Algorithm Set $\mathcal{A}$
Output: Negative Dataset $\mathcal{D}_\text{neg}$

Function `AccMetric`$(\mathcal{A}, M, d)$:
  Input: Algorithm $\mathcal{A}$, Data point d
  Output: Accuracy p
  Adopt compression algorithm $\mathcal{A}$ and LLM M to produce the response r for given prompt d.
  Return the accuracy with the response r.
End Function

Initialize $\mathcal{D}_\text{neg} = \varnothing$.
for d_i **in** $\mathcal{D}$ **do**
  p_base = `AccMetric`$(\mathcal{A}_\text{base}, d_i)$
  $negative$ = **true**
  for $\mathcal{A}_j$ **in** $\mathcal{A}$ **do**
    if `AccMetric`$(\mathcal{A}_j, d_i) \geq (1-\theta) \times p_\text{base}$ **then**
      $negative$ = **false**
    end if
  end for
  if $negative$ **then**
    Insert d_i in $\mathcal{D}_\text{neg}$
  end if
end for

5 A Suite of Tools

Given the above limitations of existing studies, we introduce a set of tools to assist `KV cache` compression.

5.1 Throughput Predictor

First, it is common for online LLM serving systems to handle various sequence lengths and batch sizes in the prefill and decoding stages. In LLM serving, all other operations (e.g., linear product) are independent of `KV cache` and attention operation. Fortunately, `KV cache` compression primarily impacts the throughput performance of the attention operations. Hence, we profile the throughput of the attention layer across various sequence lengths and batch sizes in both prefill and decoding stages. We incorporate the offline profiled throughput results into the LLM runtime predictor developed by Vidur (Agrawal et al., 2024) to inform the LLM serving systems to make appropriate scheduling decisions. Table 6 reports the accuracy of our throughput predictor for LLaMA-7B. Our observation is that it can provide above 85% prediction accuracy for various compression techniques. More experimental details about the throughput predictor can be found in Appendix F.

5.2 Length Predictor

Recent works (Zheng et al., 2023; Qiu et al., 2024) demonstrate the potential for predicting the response length in LLMs. Following this direction, we gather response length from various `KV cache` compression algorithms and em-

ploy a BERT-based classifier to predict the length generated by a given compression algorithm. Table 6 reports the prediction results of our length predictor for LLaMA-3.1-8B-instruct, which can attain above 85% accuracy across four representative KV cache algorithms. More experimental details and results about the length predictor are elaborated in Appendix G. Our length predictor can inform the online serving system to determine whether to apply compression techniques on incoming requests, thus mitigating the extended end-to-end latency.

Table 6. The prediction accuracy of our proposed tools.

Tools	FP16	KIVI	GEAR	H2O	Stream
Throughput Predictor	88.5%	88.4%	87.7%	85.8%	86.6%
Length Predictor	89.3%	95.7%	88.4%	87.8%	90.0%

5.3 Negative Sample Evaluator

Based on our empirical analysis, we set a 10% threshold to identify negative samples and compile them into a benchmark dataset. This dataset evaluates both existing and future KV cache compression techniques. Using LongBench's evaluation score, we measure the performance of four representative compression methods on LLaMA-3.1-8B-instruct, as reported in Table7. The baseline (FP16) achieves high scores, but various KV cache compression algorithms show significant drops. More details are provided in Appendix H. We recommend further research into these negative samples to better understand the impact of KV cache compression algorithms on accuracy.

To mitigate negative samples, we recommend the following solutions to realize task-specific KV cache compression techniques. First, we can adopt a lightweight model to predict the task types of input requests for LLM serving. Second, we can develop task-specific KV cache compression approaches or adopt KV cache with varying compression levels.

Table 7. The measured score of various algorithms on the negative sample benchmark dataset using LongBench's provided metric.

Task Type	Baseline	KIVI	GEAR	H2O	Stream
Summarization	31.6	24.8	23.7	24.7	24.3
Question Answering	52.0	28.8	28.7	33.8	30.4
Code	97.0	30.0	30.0	57.2	61.3

5.4 Usage of Tools: Request Router

We leverage our throughput and length predictor to explore how both tools can be used to expedite online LLM serving via request routing. Particularly, we run LLaMA-7B on four A6000 GPUs with LMDeploy, and sample 1000 requests from ShareGPT, using Poisson distributions with request per second as 10. *Baseline* refers to running LLaMA-7B with FP 16 or a given KV cache compression approach on four GPUs. It adopts a load-balancing technique to route incoming requests to a GPU with minimum memory usage. In the following three policies, we empirically use one GPU to run FP16 and three GPUs to run a given compression technique. We need to route an incoming request to an appropriate GPU. *w/ Throughput* refers to routing a request to a GPU that can yield the estimated highest decoding throughput by the throughput predictor. *w/ Length* refers to routing to a GPU, which can yield the estimated minimum response length by length predictor. *w/ Both* refers to routing to a GPU that can yield the minimum estimated end-to-end latency. The end-to-end latency is calculated by the prefilling time and a product between the estimated decoding throughput and estimated response length. We report the average end-to-end latency (seconds) for FP16 and compression techniques in Table 8. The throughput predictor speeds up the end-to-end latency by 1.18-1.48×. In contrast, the length predictor yields a speedup of 0.83-1.03×, suggesting that relying solely on the length predictor potentially compromises the latency. Combining the throughput predictor and length predictor speeds up the latency by 1.45 to 1.80×.

Table 8. Average end-to-end latency of different routing methods.

Average E2E	FP16	KIVI	GEAR	H2O	Stream
Baseline	11.4	9.1	13.4	10.6	10.3
w/ Throughput	-	7.7	9.1	8.3	8.2
w/ Length	-	10.9	13	11.2	11.3
w/ Both	-	6.3	7.4	6.9	6.6

6 Conclusion

In this paper, we present a retrospective study of KV cache compression for LLM serving. We conduct a comprehensive literature survey and empirical analysis of existing algorithms, identifying several under-explored aspects of their practical usage. Our analysis reveals key difficulties that hinder the real-world deployment of KV cache compression. We recommend dissecting the evaluation of LLM KV cache compression algorithms into three critical dimensions, including throughput, length distribution, and negative samples. We gather insights from both literature and our evaluations to design three tools aimed at facilitating the applications of LLM KV cache compression algorithms in the production environment.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. The research is supported under the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s).

REFERENCES

Issue #4: [integrate kivi into inference frameworks?]. https://github.com/jy-yuan/KIVI/issues/4. Accessed: 2025.04.

Adnan, M., Arunkumar, A., Jain, G., Nair, P., Soloveychik, I., and Kamath, P. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Agrawal, A., Kedia, N., Mohan, J., Panwar, A., Kwatra, N., Gulavani, B., Ramjee, R., and Tumanov, A. Vidur: A large-scale simulation framework for llm inference, 2024. URL https://arxiv.org/abs/2405.05465.

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Anon. Sharegpt vicuna unfiltered dataset. https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered, 2024. Accessed: 2024-06-13.

Anthropic. Claude ai, 2024. URL https://claude.ai/. Accessed: 2024-09.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*, 2024.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

BentoML. Benchmarking llm inference backends. https://bentoml.com/blog/benchmarking-llm-inference-backends. Accessed: 2025.03.

Chang, C.-C., Lin, W.-C., Lin, C.-Y., Chen, C.-Y., Hu, Y.-F., Wang, P.-S., Huang, N.-C., Ceze, L., and Wu, K.-C. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Chen, Y., Wang, G., Shang, J., Cui, S., Zhang, Z., Liu, T., Wang, S., Sun, Y., Yu, D., and Wu, H. Nacl: A general and effective kv cache eviction framework for llms at inference time. *arXiv preprint arXiv:2408.03675*, 2024.

Contributors, L. Lmdeploy: A toolkit for compressing, deploying, and serving llm. https://github.com/InternLM/lmdeploy, 2023.

Dai, J., Huang, Z., Jiang, H., Chen, C., Cai, D., Bi, W., and Shi, S. Sequence can secretly tell you what to discard. *arXiv preprint arXiv:2404.15949*, 2024.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Dong, H., Yang, X., Zhang, Z., Wang, Z., Chi, Y., and Chen, B. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. *arXiv preprint arXiv:2402.09398*, 2024a.

Dong, S., Cheng, W., Qin, J., and Wang, W. Qaq: Quality adaptive quantization for llm kv cache. *arXiv preprint arXiv:2403.04643*, 2024b.

Duanmu, H., Yuan, Z., Li, X., Duan, J., Zhang, X., and Lin, D. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*, 2024.

Feng, Y., Lv, J., Cao, Y., Xie, X., and Zhou, S. K. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference, 2024. URL https://arxiv.org/abs/2407.11550.

Flashinfer. Flashinfer: A lightweight framework for inferencing. https://github.com/flashinfer-ai/flashinfer, 2024. Accessed: 2024-10.

Fu, Q., Cho, M., Merth, T., Mehta, S., Rastegari, M., and Najibi, M. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*, 2024.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Gong, R., Yong, Y., Gu, S., Huang, Y., Zhang, Y., Liu, X., and Tao, D. Llm-qbench: A benchmark towards the best practice for post-training quantization of large language models. *arXiv preprint arXiv:2405.06001*, 2024.

Hassabis, D. and the Gemini Team. Introducing gemini: our largest and most capable ai model, 2023.

URL https://blog.google/technology/ai/google-gemini-ai. Accessed: 2024-06-07.

He, Y., Zhang, L., Wu, W., Liu, J., Zhou, H., and Zhuang, B. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *arXiv preprint arXiv:2405.14256*, 2024.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Kang, H., Zhang, Q., Kundu, S., Jeong, G., Liu, Z., Krishna, T., and Zhao, T. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023a.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023b.

Li, S., Ning, X., Wang, L., Liu, T., Shi, X., Yan, S., Dai, G., Yang, H., and Wang, Y. Evaluating quantized large language models. *arXiv preprint arXiv:2402.18158*, 2024a.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024b.

Liu, A., Liu, J., Pan, Z., He, Y., Haffari, G., and Zhuang, B. Minicache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*, 2024a.

Liu, D., Chen, M., Lu, B., Jiang, H., Han, Z., Zhang, Q., Chen, Q., Zhang, C., Ding, B., Zhang, K., et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024b.

Liu, R., Bai, H., Lin, H., Li, Y., Gao, H., Xu, Z., Hou, L., Yao, J., and Yuan, C. Intactkv: Improving large language model quantization by keeping pivot tokens intact. *arXiv preprint arXiv:2403.01241*, 2024c.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024d.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024e.

Luohe, S., Hongyi, Z., Yao, Y., Zuchao, L., and Hai, Z. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

Qiu, H., Mao, W., Patke, A., Cui, S., Jha, S., Wang, C., Franke, H., Kalbarczyk, Z. T., Başar, T., and Iyer, R. K. Efficient interactive llm serving with proxy model-based sequence length prediction. In *The 5th International Workshop on Cloud Intelligence / AIOps at ASPLOS 2024*, volume 5, pp. 1–7, San Diego, CA, USA, 2024. Association for Computing Machinery.

Ren, S. and Zhu, K. Q. On the efficacy of eviction policy for key-value constrained generative language model inference. *arXiv preprint arXiv:2402.06262*, 2024.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Shi, Z., Ming, Y., Nguyen, X.-P., Liang, Y., and Joty, S. Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction. *arXiv preprint arXiv:2409.17422*, 2024.

Tang, H., Lin, Y., Lin, J., Han, Q., Hong, S., Yao, Y., and Wang, G. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024a.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024b.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wang, Z. and Gan, S. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. *arXiv preprint arXiv:2404.04793*, 2024.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Xiao, C., Zhang, P., Han, X., Xiao, G., Lin, Y., Zhang, Z., Liu, Z., and Sun, M. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. In *First Workshop on Long-Context Foundation Models@ ICML 2024*, 2024a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Xiao, G., Tang, J., Zuo, J., Guo, J., Yang, S., Tang, H., Fu, Y., and Han, S. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024b. URL https://arxiv.org/abs/2410.10819.

Xu, Y., Jie, Z., Dong, H., Wang, L., Lu, X., Zhou, A., Saha, A., Xiong, C., and Sahoo, D. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*, 2024.

Yang, D., Han, X., Gao, Y., Hu, Y., Zhang, S., and Zhao, H. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024a.

Yang, J. Y., Kim, B., Bae, J., Kwon, B., Park, G., Yang, E., Kwon, S. J., and Lee, D. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *arXiv preprint arXiv:2402.18096*, 2024b.

Yang, S., Sheng, Y., Gonzalez, J. E., Stoica, I., and Zheng, L. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*, 2024c.

Yuan, J., Liu, H., Chuang, Y.-N., Li, S., Wang, G., Le, D., Jin, H., Chaudhary, V., Xu, Z., Liu, Z., et al. Kv cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches. *arXiv preprint arXiv:2407.01527*, 2024.

Yue, Y., Yuan, Z., Duanmu, H., Zhou, S., Wu, J., and Nie, L. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

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

Zhang, H., Ji, X., Chen, Y., Fu, F., Miao, X., Nie, X., Chen, W., and Cui, B. Pqcache: Product quantization-based kvcache for long context llm inference. *arXiv preprint arXiv:2407.12820*, 2024a.

Zhang, T., Yi, J., Xu, Z., and Shrivastava, A. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Advances in Neural Information Processing Systems*, 37:3304–3331, 2024b.

Zhang, Y., Du, Y., Luo, G., Zhong, Y., Zhang, Z., Liu, S., and Ji, R. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*, 2024c.

Zhang, Y., Gao, B., Liu, T., Lu, K., Xiong, W., Dong, Y., Chang, B., Hu, J., Xiao, W., et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024d.

Zhang, Z. and Shen, H. Zero-delay qkv compression for mitigating kv cache and network bottlenecks in llm inference. *arXiv preprint arXiv:2408.04107*, 2024.

Zhang, Z., Liu, S., Chen, R., Kailkhura, B., Chen, B., and Wang, A. Q-hitter: A better token oracle for efficient llm inference via sparse-quantized kv cache. *Proceedings of Machine Learning and Systems*, 6:381–394, 2024e.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024f.

Zheng, Z., Ren, X., Xue, F., Luo, Y., Jiang, X., and You, Y. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *arXiv preprint arXiv:2305.13144*, 2023.

Zhu, Q., Duan, J., Chen, C., Liu, S., Li, X., Feng, G., Lv, X., Cao, H., Chuanfu, X., Zhang, X., Lin, D., and Yang, C. Sampleattention: Near-lossless acceleration of long context llm inference with adaptive structured sparse attention, 2024. URL https://arxiv.org/abs/2406.15486.

A Artifact Appendix

A.1 Abstract

We provide an artifact to demonstrate that representative `KV cache` compression methods can reduce memory consumption. Current implementations (e.g., FlashAttention, PagedAttention) do not optimize for production-level LLM serving, resulting in suboptimal throughput performance. We also provide our tools to facilitate future LLM `KV cache` compression studies.

A.2 Description and Requirements

Github Repository. The codes are available in the following Github repository https://github.com/LLMkvsys/rethink-kv-compression.

Hardware dependencies. Experiments are conducted on one NVIDIA A6000 GPU.

Software dependencies. We provide a Docker image with NVIDIA GPU support for this artifact. We use CUDA 12.1, torch 2.1.2, transformers 4.43.1, and LMDeploy v6.0.1 (modified).

A.3 Setup

To install the artifact, users should clone the repository.

```
git clone git@github.com:LLMkvsys/
    rethink-kv-compression.git
cd rethink-kv-compression/
conda env create -f mlsys_environment.
    yml
conda activate lmdeploy
conda clean -a
cd benchmark_thr/src/
pip install -e .
```

We also provide a Dockerfile to help you build a docker image.

A.4 Evaluation workflow

Given the significant time and resource costs associated with length and negative sample analysis, we do not recommend executing these scripts during artifact evaluation. Instead, we suggest focusing on throughput analysis (Figure 1). We have provided scripts to facilitate the reproduction of Figure 1 using a single GPU. However, the first row in Figure 1 involves running models with HF transformers, which can be notably slow in this context. As a result, we have not included corresponding scripts for this specific part. **We recommend users follow the README as it provides more detailed explanations.**

A.5 Major Claims

The KV cache compression methods show negative computational efficiency in certain scenarios of batch size, sequence length. Most KV cache compression methods except Gear show advantages when serving requests with a heavy KV cache.

A.6 Experiments

It might take one hour to finish the following experiments. We provide scripts to measure the prefill and decoding throughput of quantization-based approaches across various sequences with a fixed batch size. The benchmarking logs are stored in folders *0_quant_normal_logs* and *0_quant_long_logs*, respectively.

```
cd benchmark_thr/src
bash 0_quant_normal_logs/
    batch_eval_quant_normal_fixbsz.sh
bash 0_quant_long_logs/
    batch_eval_quant_long_fixbsz.sh
```

The following scripts are to measure the prefill and decoding throughput of sparsity-based approaches across various sequences with a fixed batch size. The benchmarking logs are stored in folders *0_sparse_normal_logs* and *0_sparse_long_logs*, respectively.

```
cd benchmark_thr/src
bash 0_sparse_normal_logs/
    batch_eval_sparse_normal_fixbsz.sh
bash 0_sparse_long_logs/
    batch_eval_sparse_long_fixbsz.sh
```

We provide scripts to measure the prefill and decoding throughput of quantization-based approaches across various batch sizes with a fixed sequence length. The benchmarking logs are stored in folders *0_quant_normal_logs* and *0_quant_long_logs*, respectively.

```
cd benchmark_thr/src
bash 0_quant_normal_logs/
    batch_eval_quant_normal_fixlen.sh
bash 0_quant_long_logs/
    batch_eval_quant_long_fixbsz.sh
```

We provide scripts to measure the prefill and decoding throughput of quantization-based approaches across various batch sizes with a fixed sequence length. The benchmarking logs are stored in folders *0_sparse_normal_logs* and *0_sparse_long_logs*, respectively.

```
cd benchmark_thr/src
bash 0_sparse_normal_logs/
    batch_eval_sparse_normal_fixlen.sh
bash 0_sparse_long_logs/
    batch_eval_sparse_long_fixbsz.sh
```

A.7 Experiments Results

We provide the following plotting scripts to generate Figures 1 (e)–(i), respectively. The resulting figures are saved in the folder *demo_figs/*. To account for system noise, we recommend that users verify the presence of two key findings: (1) certain KV cache compression methods fail to outperform FP16 baselines; (2) the decoding throughput advantages of KV cache compression techniques become more pronounced in scenarios with heavy KV cache usage.

```
cd benchmark_thr/src
# Figure 1 (e) & (i)
python 0_plot/plot_normal_fixbsz.py
# Figure 1 (f) & (j)
python 0_plot/plot_normal_fixseqlen.py
# Figure 1 (g) & (k)
python 0_plot/plot_long_fixbsz.py
# Figure 1 (h) & (i)
python 0_plot/plot_long_fixseqlen.py
```

A.8 Additional Experiments

We conduct length analysis and negative analysis and provide our tools for your reference. Since this process requires substantial GPU resources, we offer precomputed, cached results within our prepared environment. The cached results comprise the generated responses and associated length and evaluation metric score.

First, we provide cached results to reproduce Figure 3.

```
cd benchmark_len/
python -u plot_kde_shift_dist.py
```

This script yields the distribution of response length difference across various compression algorithms. The results are saved in the current directory. The users can observe that with the increase of the compression ratio, the distribution of response difference flattens, and more samples experience verbose response.

Second, we provide cached results to reproduce Figure 5.

```
cd benchmark_neg/
python -u 0_ratio_vs_no_negative.py
```

This script outputs the figures in which the number of negative samples changes with the threshold. The results are saved in the current directory. The users can find that there are many negative samples even with a threshold of 10%, indicating the fragility of compression algorithms.

B Evaluation Details

B.1 Dataset

ShareGPT. We select a subset of requests from ShareGPT (Anon, 2024) to conduct the experiments of response length difference distribution. We refer to the benchmark code in vLLM[3] to sample 1, 000 requests. Due to time and resource constraints, we set the maximum number of generation tokens as 1024 in the evaluation. We also truncate contexts for input prompts that exceed the model's maximum length allowance to ensure they fit within the model's capacity.

LongBench. LongBench (Bai et al., 2023) is a task for long context understanding that covers key long-text application scenarios, including multi-document QA, single-document QA, summarization, few-shot learning, code completion, and synthetic tasks. We keep strictly the evaluation metrics and settings in their released codebase [4] to ensure fair assessments.

B.2 Models

LLaMA Family. The LLaMA family, developed by Meta using a high-quality corpus, is widely favored by researchers working on `KV cache` compression algorithms. Many choose LLaMA models to evaluate the effectiveness of their methods. In our performance evaluation, we cover LLaMA-2-7B, LLaMA-2-13B, and LLaMA-2-70B to emphasize the advantages of `KV cache` due to their exorbitant GPU memory consumption of `KV cache`. Additionally, LLaMA-3.1-8B, known for generating high-quality responses and excelling in long-context tasks, is used in our length distribution and negative sample analysis.

Mistral Family. Similarly to the LLaMA family, many models from the Mistral family are used to demonstrate the benefits of `KV cache` algorithms. Mistral models incorporate grouped-query attention (GQA) for faster inference and are renowned for their exceptional performance. In our length difference and negative sample analysis, we utilize Mistral-7B-v0.1 to obtain relevant experimental results.

B.3 Algorithms

KIVI. KIVI (Liu et al., 2024e) is a notable quantization algorithm for `KV cache` compression, specializing in per-channel quantization for key tensors and per-token quantization for value tensors. We utilize their official implementation [5]. The critical hyperparameters in KIVI are group size G and the residual length R. G refers to the number of

[3] https://github.com/vllm-project/vllm/blob/main/benchmarks/benchmark_serving.py

[4] https://github.com/THUDM/LongBench

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

channels that are grouped for quantization in the key cache, while R controls the number of most recent tokens that are kept in full precision. Following the paper's recommendations for achieving optimal performance, we have set them to $G = 32, R = 128$.

GEAR. GEAR (Kang et al., 2024) is a typical quantization error mitigation algorithm. We took their open-source code[6]. The key parameters of GEAR are sparsity ratio s and rank r. Specifically, s specifies the number of retained full-precision outlier values. r controls the richness of the low-rank approximation matrix, which recovers the model's ability from quantization errors. In line with the default settings in the official codebase, we set $s = 2\%, r = 2\%$.

StreamingLLM. StreamingLLM (Xiao et al., 2023) is an attention sparsity-based cache eviction algorithm. It retains only a limited number of initial and most recent tokens. The key parameters for controlling the sizes of the initial and recent tokens are set to 64 and 448, respectively, resulting in a total cache size of 512.

H2O. H2O (Zhang et al., 2024f) is another widely-used cache eviction algorithm that dynamically calculates and refreshes the `KV cache`. The parameters for the heavy hitter oracle token size and the recent size are configured to 64 and 448, respectively, with a total cache size of 512.

B.4 LLM Serving Engine.

Transformers Library. We directly use Torch 2.1.2 and Transformers 4.43.1 to measure the throughput performance.

FlashAttention. FlashAttention[7] can fully exploit the GPU resources to realize fast and memory-efficient attention operation. In our throughput evaluation, we measure the throughput performance of TRL+FA by enabling FlashAttention 2.5.6 in the transformers library.

LMDeploy. LMDeploy[8] allows LLM developers to compress, deploy, and serve various LLMs. It naturally supports the functionality of PagedAttention and FlashAttention. We implement various `KV cache` algorithms based on LMDeploy v6.0.1. We chose LMDeploy for three reasons.

First, LMDeploy stands out by implementing more efficient quantization kernels than vLLM. This results in superior performance in KV cache compression approaches compared to vLLM, despite the significant attention vLLM has garnered. Note that the primary focus of our paper is on KV cache compression approaches, with a particular emphasis on quantization. A prior benchmark study conducted by BentoML (BentoML) uncovers that LMDeploy obtains the best throughput performance with 4-bit quantization.

Second, LMDeploy offers a better way for faster development of KV cache compression algorithms than vLLM. The author of KVI has stated the challenges of integrating the KIVI algorithm into vLLM as early as April 2024 (kiv). As of now, there has been no significant progress on this front.

Third, our conclusions, except for Observation 2, do not pertain to any specific serving features of the inference engines. Consequently, they remain independent of the LLM inference engine used. For Observation 2, our objective is to explore the impact of KV cache compression methods like sparsity and quantization on popular serving features (e.g., Page Attention, Flash Attention) rather than focusing on any particular serving engine. As long as the selected inference engines support the efficient implementation of the necessary serving features (Page Attention, FlashAttention), it will not affect Observation 2.

B.5 Hardware Environment.

Our evaluation experiments are conducted on a GPU node with four NVIDIA A6000 GPUs interconnected via NVLink and powered by an Intel Xeon Gold 6326 CPU at 2.90 GHz.

C More results of Throughput Analysis

We evaluate throughput performance on a GPU node with four NVIDIA A6000 GPUs interconnected via NVLink and powered by an Intel Xeon Gold 6326 CPU at 2.90 GHz. We exclude the initialization overhead and average the throughput performance over three times for fair comparison. We add more experiments to demonstrate the generality of our statement in the throughput analysis as follows.

First, we measure the prefill and decoding throughput on TRL, TRL+FA, and LMD using Mistral-7B and LLaMA-13B, depicted in Figure 8 (a-b) and Figure 10 (a-b), respectively. The relative speedup of the StreamingLLM algorithm in the decoding throughput varies across LLMs and serving techniques, as shown in Figure 8 (c-d) and Figure 10 (c-d). The high speedup from TRL does not mean the significant speedup benefits.

Second, we measure the prefill and decoding throughput on LMD with various batch and prompt lengths in Figure 8 (e-h) and Figure 10 (e-l). We have observed that these Large Language Models (LLMs) show negative speedup in certain prompt lengths and batch sizes, which is consistent with the statement mentioned in Section 4. However, the prompt lengths and batch sizes that lead to this disadvantage vary among different LLMs. Worth noticing that in Figure 10, we

[6] https://github.com/opengear-project/GEAR

[7] https://github.com/Dao-AILab/flash-attention

[8] https://github.com/InternLM/lmdeploy/tree/main

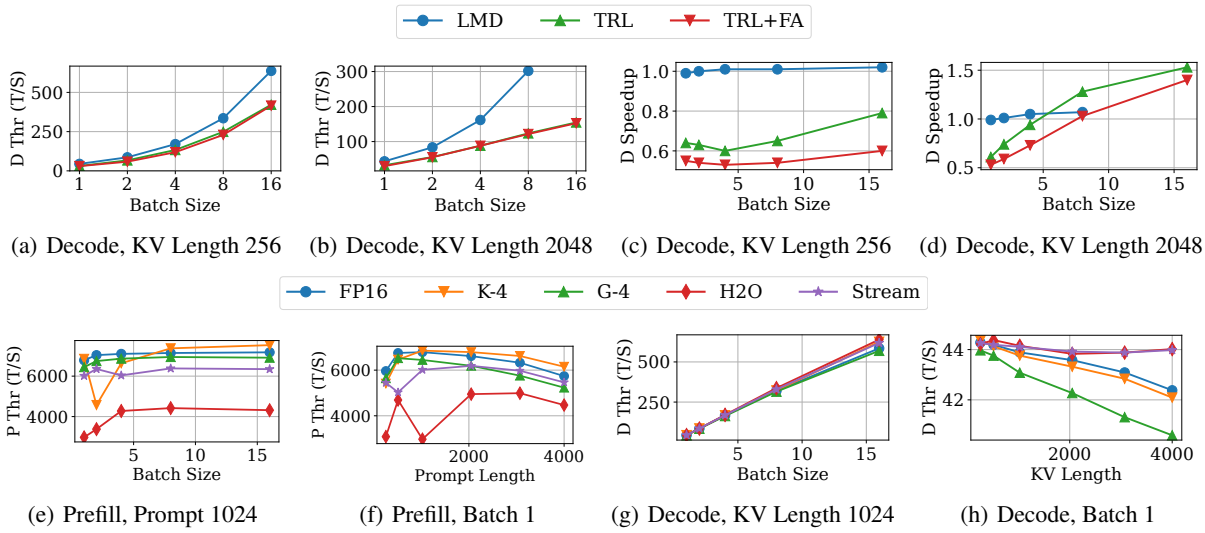


Figure 8. Throughput analysis of **Mistral-7B** (a-b) The FP16 decoding throughput on TRL (with and without FlashAttention) and LMDeploy (LMD). (c-d) The speedup of the KIVI-4bit algorithm on TRL and LMD. (e-h) The prefill and decoding throughput for inputs of moderate size.

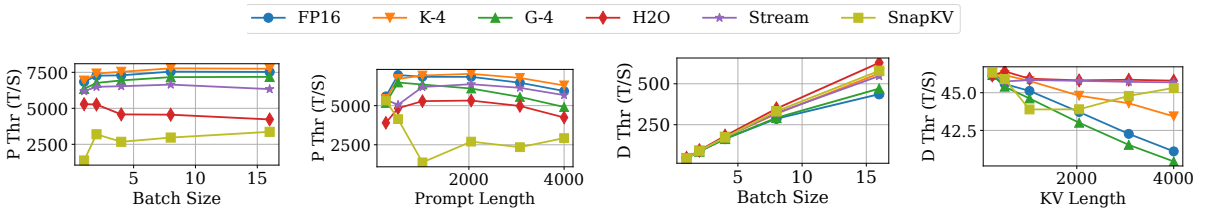


Figure 9. Throughput analysis of **LLaMA-7B**, with `KV cache` compression algorithm SnapKV (Li et al., 2024b) integrated.

omit the throughput information for the KIVI-4 algorithm due to the out-of-memory issue when evaluating on LLaMA-13B with a single A6000 GPU.

Third, we present additional findings on tensor parallelism in Figures 11, 12, 13, and 14. Performance improvements with larger tensor parallelism (TP) are notably evident during the prefill stage for various compression methods. However, larger TP does not confer significant benefits in the decoding stage when the batch size is small. Our observations indicate that the throughput advantages derived from `KV cache` compression typically become more pronounced under heavy evaluation settings (e.g., batch size, KV length, and model size).

D More Results of Length Analysis

First, we outline the configurations of the compression algorithms in Section 4.3. For text generation, we fix the temperature as 1 for the FP16 baseline and compression methods. We also vary T to assess the impact of length differences from the hyperparameter T in Table 5. For quantization-based methods, we only vary the quantization bits for KIVI and GEAR. For sparsity-based methods, we only vary the `KV cache` length for StreamingLLM and H2O. All other compression-related configurations remain consistent with those detailed in Appendix B.3.

Second, we supply more experimental results on Mistral-7B to demonstrate the generality of our statement in **Observations 3 and 4**, respectively. Particularly, we perform a similar experimental analysis as Table 5 on Mistral-7B and show the ratio (%) of samples experiencing response length variations induced by temperature and `KV cache` compression in Table 9. Similar to the LLaMA model, `KV cache` compression shows a clear tendency to produce verbose responses in Mistral-7B. We also repeat the experiments in Figure 4 and show the results of Mistral-7B in Figure 15. The impact of the compression ratio on the response length remains consistent between the LLaMA and Mistral. We also measure the end-to-end latency for various compression algorithms on Mistral-7B, as shown in Figure 16. Our observation is that the latency benefits of `KV cache` compression are not prominent, and the verbose response length should be accounted for performance measurement of various `KV cache` compression.

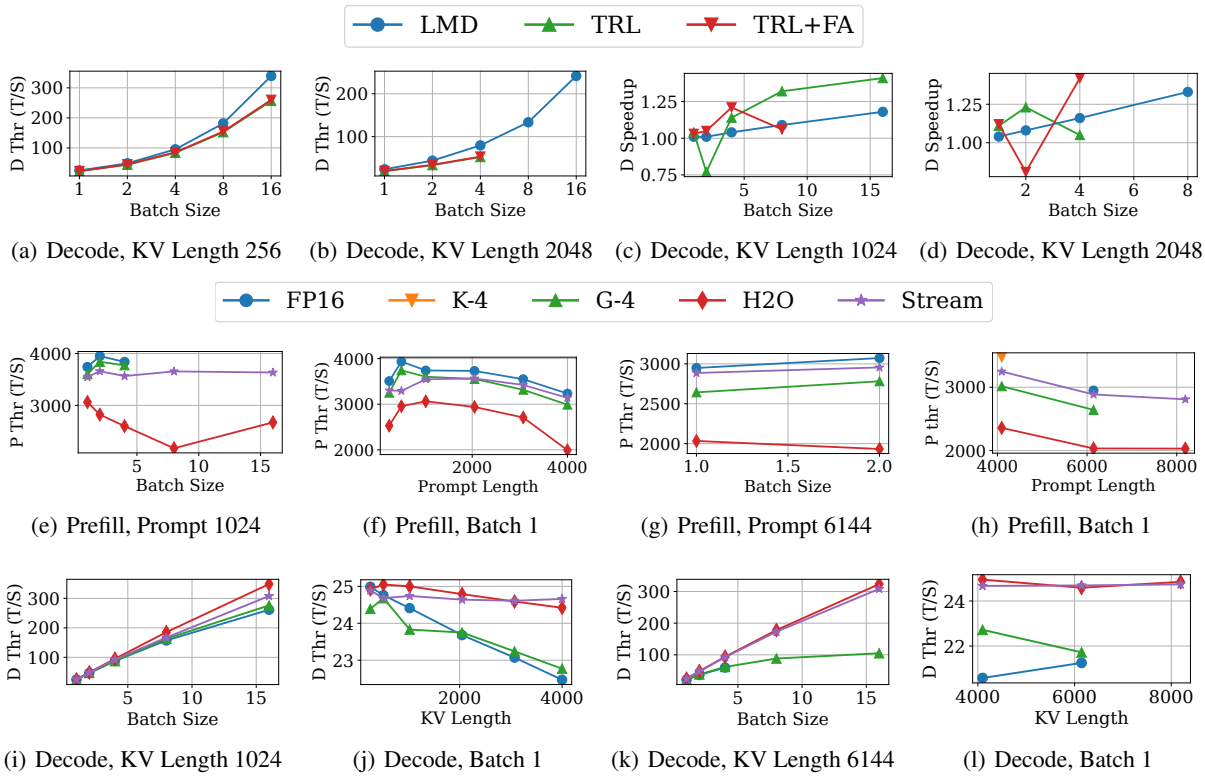


Figure 10. Throughput analysis of **LLaMA-13B**: (a-b) The FP16 decoding throughput on TRL (with and without FlashAttention) and LMDeploy (LMD). (c-d) The speedup of the StreamingLLM algorithm on TRL and LMD. (e-h) The prefill throughput for various sizes of inputs. (i-l) The decoding throughput for various sizes of inputs.

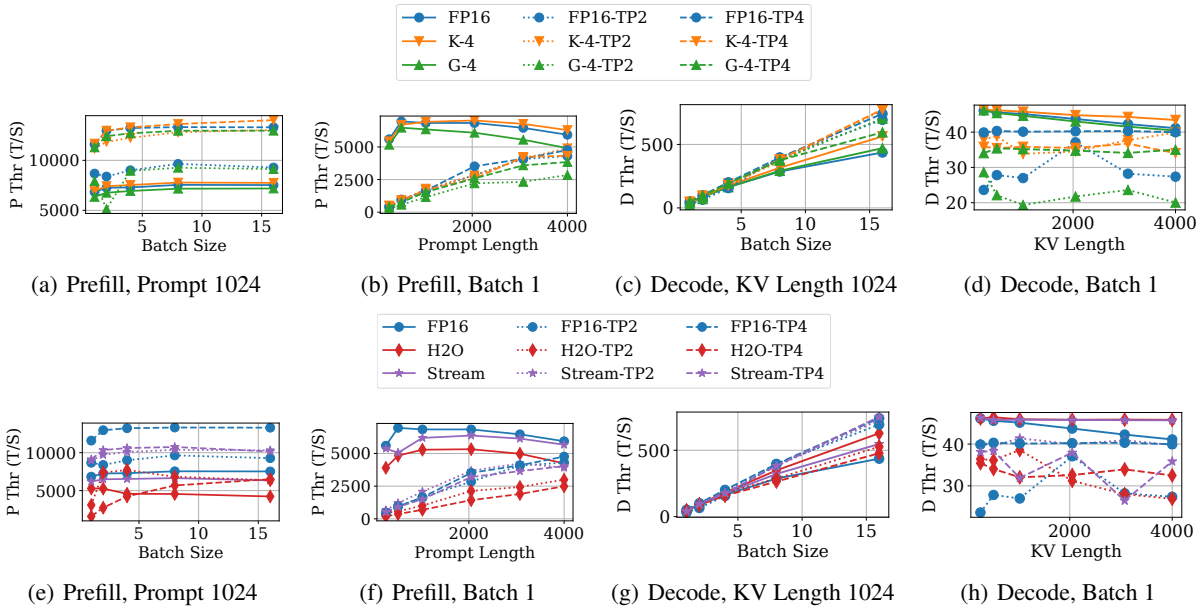
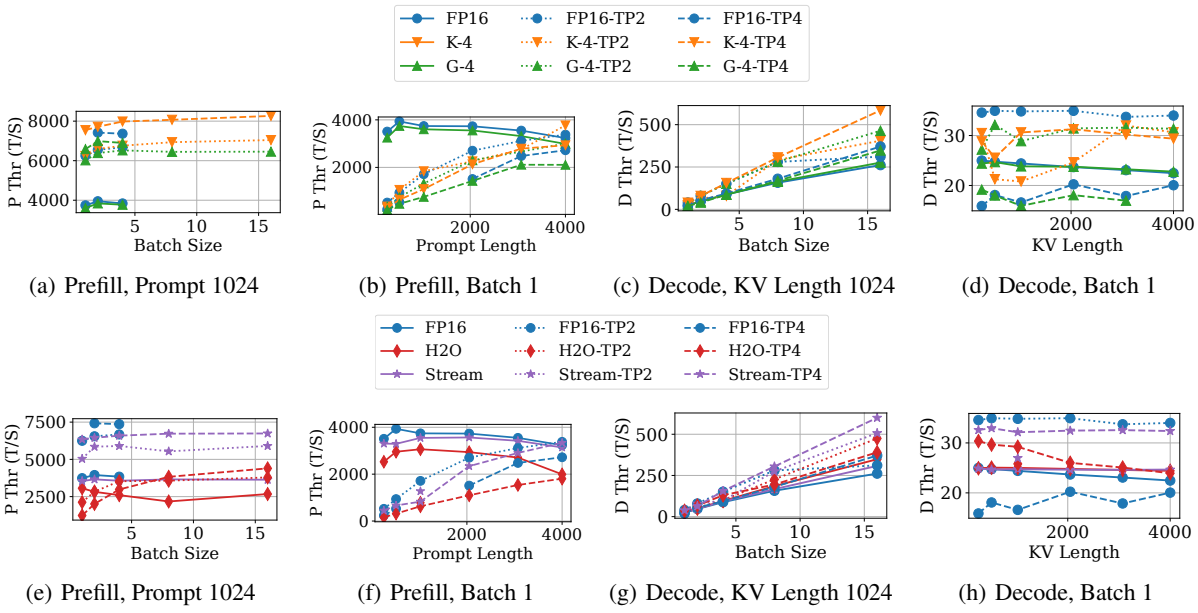


Figure 11. Throughput analysis of **LLaMA-7B**, with different tensor parallelism configurations. (a-d) The throughput of quantization-based methods. (e-h) The throughput of sparsity-based methods.

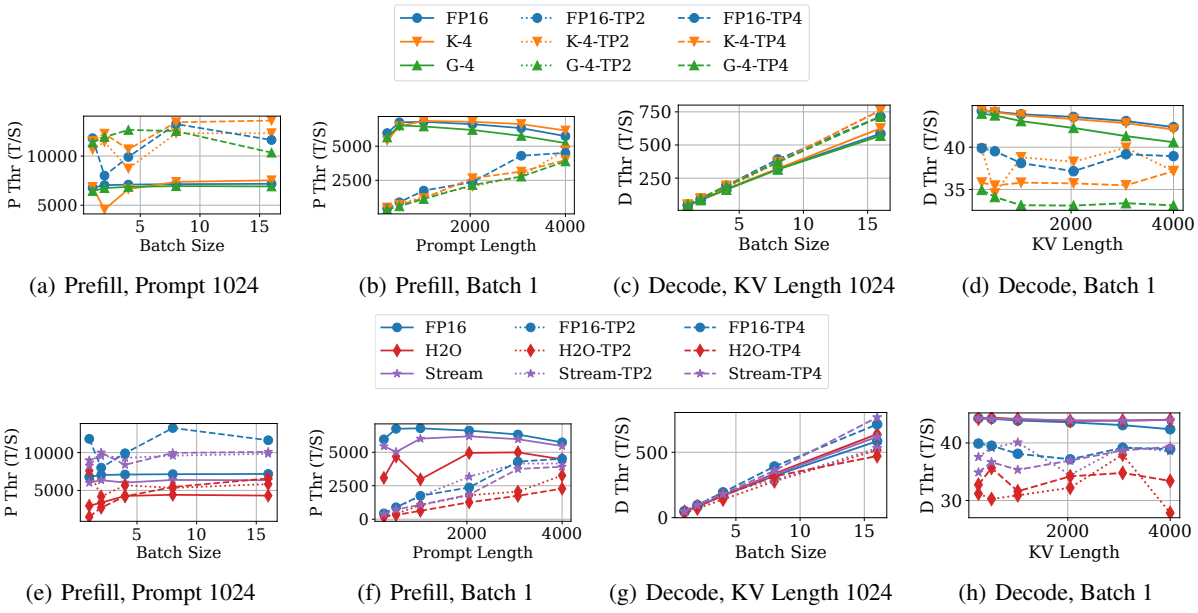
Third,



(a) Prefill, Prompt 1024 (b) Prefill, Batch 1 (c) Decode, KV Length 1024 (d) Decode, Batch 1

(e) Prefill, Prompt 1024 (f) Prefill, Batch 1 (g) Decode, KV Length 1024 (h) Decode, Batch 1

Figure 12. Tensor parallelism analysis of **LLaMA-13B**. (a-d) The throughput of quantization-based methods. (e-h) The throughput of sparsity-based methods.



(a) Prefill, Prompt 1024 (b) Prefill, Batch 1 (c) Decode, KV Length 1024 (d) Decode, Batch 1

(e) Prefill, Prompt 1024 (f) Prefill, Batch 1 (g) Decode, KV Length 1024 (h) Decode, Batch 1

Figure 13. Tensor parallelism analysis of **Mistral-7B**. (a-d) The throughput of quantization-based methods. (e-h) The throughput of sparsity-based methods.

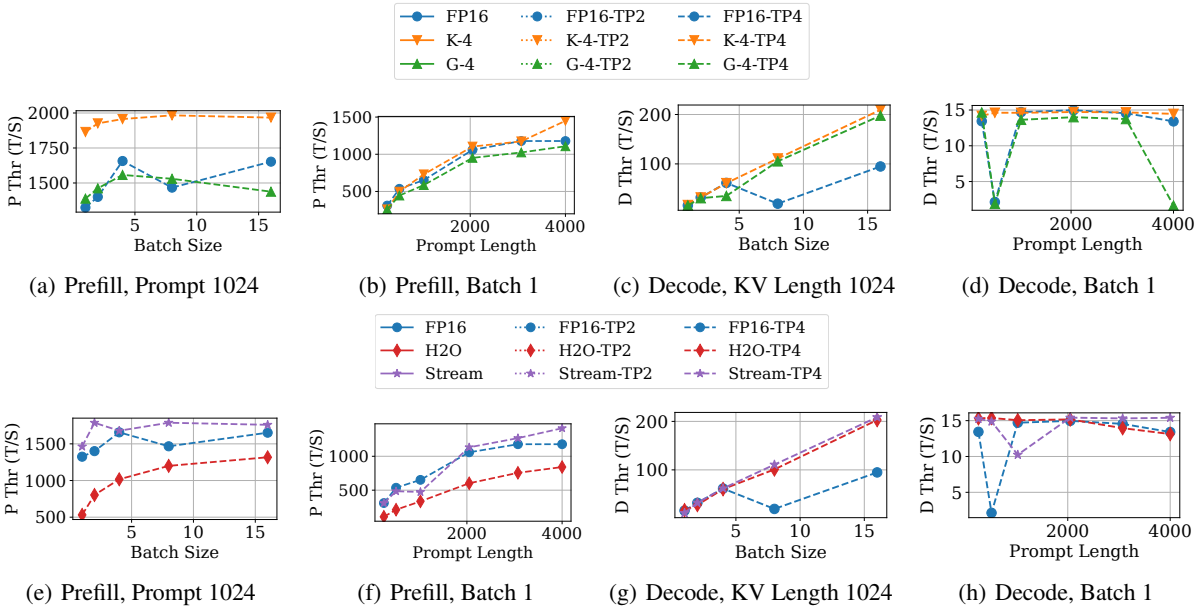
Table 9. The results of length analysis similar to Table 5, but measured on Mistral-7B.

Metric	T=0.9	T=1.1	KIVI	GEAR	H2O	Stream
% of samples D of which $\geq 50\%$	45.1%	45.9 &	2.8%	0.8%	11.0%	17.3%
% of samples D of which $\leq -50\%$	17.7%	20.0 %	44.9%	49.4%	14.3%	16.3%

E More Results of Negative Sample Analysis

First, the detailed task description of the LongBench used in the negative sample analysis can be found in https://huggingface.co/datasets/THUDM/LongBench#task-description. It contains the detailed task description of the LongBench dataset, including the task name, task type, evaluation metric, and average length. We use the corresponding task type to collect the number of negative samples.

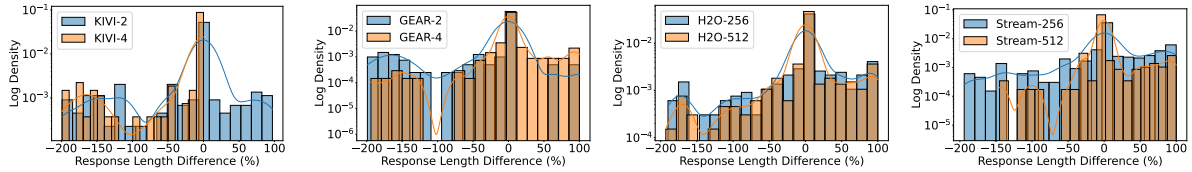
Section 3.2 suggests that compression algorithms excel in proceeding short prompt lengths with no accuracy loss. Thus, we use LongBench and Llama-3.1-8B-instruct to con-



(a) Prefill, Prompt 1024 (b) Prefill, Batch 1 (c) Decode, KV Length 1024 (d) Decode, Batch 1

(e) Prefill, Prompt 1024 (f) Prefill, Batch 1 (g) Decode, KV Length 1024 (h) Decode, Batch 1

Figure 14. Tensor parallelism analysis of **LLaMA-70B**. (a-d) The throughput of quantization-based methods. (e-h) The throughput of sparsity-based methods.



(a) KIVI (b) GEAR (c) H2O (d) StreamingLLM

Figure 15. The Mistral-7B's distribution of response length difference across different compression algorithms.

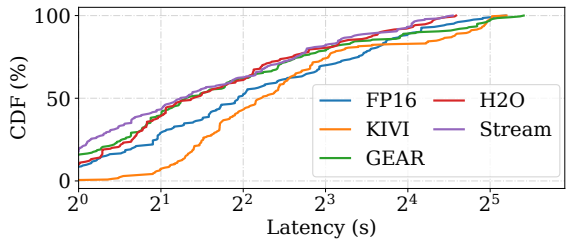


Figure 16. The Mistral-7B's CDF of the end-to-end latency (seconds) of various algorithms.

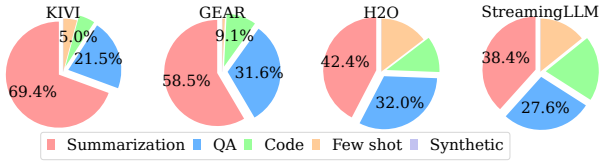
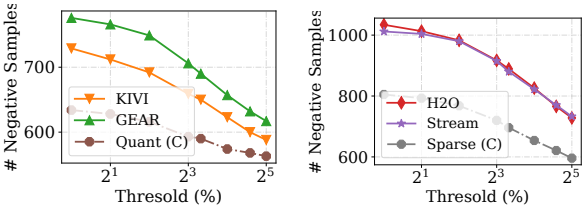


Figure 18. The pie chart details the proportion of negative samples over task types across varying compression algorithms on Mistral.



(a) Quantization (b) Sparsity

Figure 17. The threshold (x-axis) versus the number of negative samples (y-axis) for quantization-based (a) and sparsity-based (b) methods. The experimental results are measured on Mistral-7B.

duct negative sample analysis. We assess the average scores of LLaMA-3.1-8B-instruct for baseline, KIVI, GEAR, H2O, and StreamingLLM on the LongBench test dataset are 41.2, 41.3, 40.9, 39.1, and 38.9, respectively. We also cover more experimental results about negative sample analysis on Mistral-7B in Section 3.2. In Mistral-7B, the average scores for baseline, KIVI, GEAR, H2O, and StreamingLLM on the LongBench test dataset are 33.3, 33.4, 33.4, 31.8, and 30.4, respectively. First, we vary the threshold in Algorithm 1 to uncover the relationship between the threshold and the number of negative samples on Mistral-7B, as shown in Figure 15. We conclude that the minor accuracy loss from compression algorithms does not indicate that each sample

experiences a minor performance loss. It is not easy to eliminate the existence of negative samples. Second, we present the sensitivity of task types to `KV cache` compression in Figure 18 on Mistral. Similar to LLaMA, performing `KV cache` compression on Mistral-7B considerably affects the accuracy performance on summarization and QA tasks. Overall, the experimental results further reinforce our statement in Section 4.4.

Table 10. The accuracy of the length predictor for Mistral-7B.

Tools	FP16	KIVI	GEAR	H2O	Stream
Length Predictor	92.6%	92.3%	88.8%	92.8%	89.5%

F THROUGHPUT PREDICTOR

We use Vidur's released code[9] to realize the throughput predictor. The runtime time information of each operator in LLMs is profiled on A6000 NVIDIA RTX. The key difference between different compression algorithms and the FP16 baseline hinges upon the attention operation. Hence, apart from attention operators, all other operators are reused among different `KV cache` compression algorithms. We enumerate various combinations of batch sizes, sequence lengths, and stages to attain ample profiled runtime speed information for LLaMA-7B and Mistral-7B. Vidur provides the implementation code to construct and optimize the throughput predictor. We define the accuracy as $(1 - \frac{|T^{\text{pred}} - T^{\text{gt}}|}{T^{\text{gt}}}) \times 100\%$.

G LENGTH PREDICTOR

We collect the response length information from ShareGPT to synthesize the response length dataset. To account for the long-context prompt, we choose LongFormer with a maximum sequence size of 4096. We set the input of the length predictor as the input response and the target of the length predictor as the ratio between the response length and the prompt length. We define the accuracy as $(1 - \frac{|L^{\text{pred}} - L^{\text{gt}}|}{L^{\text{gt}}}) \times 100\%$. Table 6 has reported the prediction results on LLaMA3-8B. We include the prediction results on Mistral-7B in the second row of Table 10. Overall, the bert-based length predictor can deliver accurate response length prediction for LLaMA and Mistral models.

Table 11. The measured score of various algorithms evaluated on the negative sample benchmark dataset and Mistral-7B using LongBench's provided metric.

Task Type	Baseline	KIVI	GEAR	H2O	Stream
Summarization	27.2	15.2	16.6	15	11.1
Question Answering	26.8	17.6	16.4	18.0	15.4
Code	90.8	47.5	47.5	64.7	59.6

[9] https://github.com/microsoft/vidur

H PERFORMANCE ON NEGATIVE BENCHMARK

We use the Mistral-7B and report the corresponding measured score on the negative sample benchmark dataset in Table 11.