

AdaCode: Adaptive Lossless Compression for BFloat16 KV Cache

Changshi Li¹, Peiyu Liu², Liang Chen³, Yuyao Zhai⁴, Jerry Zhijian Yang⁵

¹School of Computer Science, Wuhan University,

²University of International Business and Economics,

³Peking University, ⁴Beijing Technology and Business University,

⁵School of Mathematics and Statistics, Wuhan University

lichangshi@whu.edu.cn, liupeiyustu@163.com, clandzyy@pku.edu.cn,

zhaiyuyao@stu.pku.edu.cn, zjyang.math@whu.edu.cn

Abstract

The Key-Value (KV) cache has become a critical memory bottleneck that significantly hinders the deployment of large language models, especially under long-context inference. Existing approaches either introduce non-negligible errors through lossy compression, or are tightly coupled to specific tasks or model structures, limiting their general applicability. In this work, we first identify that the exponent values in BFloat16 KV caches are highly concentrated, and further prove theoretically that such concentration is inherent to LLMs, ensuring the broad generalizability of our method. Leveraging this insight, we propose AdaCode, a bit-exact lossless compression framework. The core idea is to encode frequent exponents with a short fixed-length code while storing rare outliers explicitly. AdaCode consists of two key components: (1) a minimal-cost optimization that selects the optimal code length and exponent range by balancing the storage overhead of short codes against that of outliers, complemented by a lightweight online adjuster to handle occasional distribution shifts without GPU overhead; and (2) a GPU-friendly byte-aligned packing format together with offline calibration for parameter precomputation. Experiments show that AdaCode achieves an average compression ratio of 71.78% on diverse LLMs and 70.26% on vision-language models, with bit-exact identical recovery and no accuracy loss, effectively alleviating the KV cache memory bottleneck. The codes are available at <https://github.com/lichangshiwhu/Adacode>.

1 Introduction

The deployment of large-scale transformer models is critically constrained by the memory footprint of the Key-Value (KV) cache during autoregressive inference, especially for long-context generation. While model-side efficiency has been extensively studied (Guan et al., 2026; Lin et al., 2025), the

dynamic and input-dependent nature of the KV cache presents a distinct and pressing challenge, as its size grows linearly with both batch size and sequence length. This often limits practical context lengths or necessitates costly offloading strategies, creating a significant bottleneck for real-world applications.

Common approaches to mitigate this bottleneck primarily involve quantization (Zhao et al., 2026; Liu et al., 2024b; Li et al., 2025b; Sharma et al., 2024; Liu et al., 2024a; Badri and Shaji, 2023). While quantization of the KV cache effectively reduces its memory footprint, it inherently introduces approximation errors, potentially degrading output quality and predictability (as shown in Table 2). In scenarios where the model’s outputs must be bit-exact identical to the original uncompressed run, such as performing reasoning that requires exact reproducibility, even minor approximation errors are unacceptable. In contrast, lossless compression which preserves exact values has seen limited application to the dynamic KV cache. Existing lossless methods (Han et al., 2015; Hao et al., 2024; Hershcovitch et al., 2024; Zhang et al., 2025) are predominantly designed for static model weights and lack the efficient, low-latency compression and decompression routines required for lossless KV cache compression in high-throughput, iterative autoregressive generation on GPUs.

To overcome these limitations, we examine the numerical representation of KV caches and uncover a key structural property: the distribution of exponent values in the BFloat16 format is highly concentrated. This observation suggests that KV caches contain substantial redundancy in their exponent representation, which opens the door to lossless compression. However, turning this insight into a practical compression method is far from trivial. It raises two fundamental challenges:

- (i) *How to determine an appropriate coding length that efficiently represents the dominant ex-*

ponent range without sacrificing precision?

- (ii) *How to make such a coding scheme compatible with efficient LLM inference?*

We propose AdaCode (Adaptive Code), a bit-exact lossless KV cache compression framework that directly addresses these two challenges. The core idea is to encode frequent exponent values with a short fixed-length code while storing rare outliers explicitly, thereby preserving exact numerical fidelity. First, to achieve significant memory reduction, we formulate a minimal-cost optimization problem that selects the optimal code length and exponent range by balancing the storage overhead of short codes against that of outliers. Because distribution shifts during inference can degrade compression ratio, we further design a lightweight online adjuster that handles occasional shifts without introducing GPU overhead. Second, to support efficient KV cache access during inference, AdaCode employs a GPU-friendly byte-aligned packing format together with an offline calibration phase that precomputes compression parameters. Together, these components enable AdaCode to achieve high compression efficiency with negligible runtime overhead, making lossless KV cache compression practical for long-context LLM inference.

Extensive experiments demonstrate the effectiveness and practicality of AdaCode. Across a wide range of large language models and tasks, AdaCode achieves an average 71.78% reduction in KV cache size with bit-exact reconstruction. Moreover, thanks to its GPU-friendly design, AdaCode maintains high inference throughput, outperforming CPU offloading and the existing lossless compression baseline. We further show that AdaCode generalizes beyond text-only models and remains effective for vision language Models, highlighting its robustness across different modalities and deployment scenarios.

2 Related Works

2.1 Quantized-based KV Cache Compression

The KV cache enables long-context inference but is a major memory bottleneck (Li et al., 2025a). Quantization is a widely adopted solution since it directly operates on numerical values and is independent of model architecture (Zhao et al., 2026; Liu et al., 2024b). Quantization-based KV cache methods fall into several branches. A common line of work applies uniform low-bit quantization to keys and values, mapping float-type tensors into

INT8/INT4 or similar low-precision formats, often with per-token or per-channel affine mappings (Liu et al., 2024c; Hooper et al., 2024). With structured techniques, other works reach more extreme bitwidth regimes. For example, CommVQ (Li et al., 2025b) employs additive vector quantization, MiniKV (Sharma et al., 2024) adapts system co-design to balance aggressive 2-bit quantization with inference performance. Specifically, DecoQuant (Liu et al., 2024a) addresses outliers and numerical irregularities in the KV cache to improve quantization efficiency. While effective, all these quantization strategies are lossy and introduce approximation errors into the attention computation. This motivates our investigation of lossless compression of KV cache.

2.2 Lossless Compression Techniques for Neural Networks

Lossless compression has been extensively studied in traditional data and file compression, yet its application to neural networks remains relatively limited. Most existing work in this area focuses on coding-based techniques that exploit statistical redundancy in numerical representations rather than modifying model architectures. Early work such as Deep Compression (Han et al., 2015) introduced entropy coding, including Huffman coding, as part of a broader compression pipeline for convolutional neural networks. However, entropy coding in these approaches is applied on top of pruning and quantization, making the overall framework inherently lossy. More recently, several studies have revisited lossless encoding for LLMs. For example, ZipNN (Hershcovitch et al., 2024) applies structure-aware entropy coding to compress model weights and checkpoints, achieving substantial size reduction for large models, which is particularly beneficial for model storage and transmission. In a complementary direction, DFloat11 (Zhang et al., 2025) and Neuzip (Hao et al., 2024) bring dynamic encoding into the inference pipeline by using entropy-coded Float formats together with custom GPU kernels, significantly improving inference throughput while preserving numerical fidelity. Despite these advances, existing lossless methods focus primarily on static model parameters, which only require decompression during inference. However, existing lossless encoding methods designed for model parameters cannot be directly applied to this setting, and there has been little prior work on lossless compression specifically tailored to KV

cache representations.

3 Characteristics of BFloat16 Exponent Distributions

Our method is motivated by empirical observations of exponent distributions in BFloat16 KV caches of LLMs. We first review the BFloat16 format. BFloat16 (Brain Floating Point 16) is a 16-bit floating-point format that has become the default numerical representation for training and inference in many contemporary LLMs. Its adoption is primarily driven by a design trade-off that preserves the dynamic range of a 32-bit float (FP32) while halving the storage and bandwidth requirements. The format consists of 1 sign bit, 8 exponent bits, and 7 mantissa bits. A BFloat16 value is decoded as:

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-127} \times (1.\text{mantissa}_2), \quad (1)$$

where mantissa_2 denotes the binary fractional part interpreted as a value in $[0, 1)$. Our compression method is specifically designed around the properties of the exponent field, guided by the following three key observations.

Exponent Distribution is Highly Concentrated.

We first analyze the distribution of exponent values in the KV cache. Our results show that exponent values are not uniform but concentrated within a narrow range. Figure 1 illustrates this distribution on the C4 dataset (Raffel et al., 2019) using the Qwen3-4B model. For visual clarity, we show only the concentrated range, as the probability mass outside this range is nearly zero. The exponent distribution is highly imbalanced: the majority of values cluster around a central point, while other exponent values occur with negligible frequency. This concentration motivates a storage strategy that allocates a compact representation (a short code) to the prevalent central values and handles sparse outliers individually. In contrast, the sign and mantissa distributions are relatively flat across multiple models. Further details are provided in the *BFloat16 Distribution of KV Cache* paragraph of Appendix A.6.

Prior work has examined the distribution of KV cache values. For instance, Kaul et al. (2025) observed that KV cache (transformers’ activations) follows a kurtosis distribution, while Cai et al. (2025) modeled it as a normal distribution. In this work, we characterize the KV cache distribution using an α -stable distribution, which is a more general family that includes both normal distributions

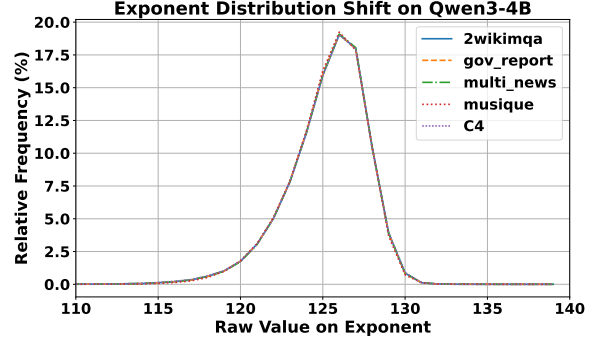


Figure 1: Exponent distribution shift for Qwen3-4B across five datasets.

and distributions with excess kurtosis. The probability of exponent values is given as follows.

Theorem 3.1 (Exponent concentration for α -stable weights). *Let $X \sim S(\alpha, 0, \gamma, 0)$ be a symmetric α -stable random variable with $\alpha \in (0, 2]$, scale $\gamma > 0$, and location zero. Denote its probability density by f and its cumulative distribution function by $F(x) = \mathbb{P}(X \leq x)$. For the BFloat16 format, set $r = 2 - 2^{-7}$ and let the stored exponent $e \in \{1, \dots, 254\}$ correspond to the absolute-value interval $[s, rs)$ with $s = 2^{e-127}$. Then the exponent probability mass function is*

$$P(e) = 2[F(rs) - F(s)], \quad (2)$$

where $s = 2^{e-127}$.

Maximizer and scale invariance. $P(e)$ attains its maximum at $\text{exponent}(s)$

$$e^* = 127 + \log_2(\gamma t^*), \quad (3)$$

where $t^* > 0$ is a root of the scale-free equation $rf_0(rt) = f_0(t)$. The peak probability

$$P(e^*) = 2[F_0(rt^*) - F_0(t^*)] \quad (4)$$

depends only on α and the format constant r ; it is independent of γ .

Concentration. The distribution is sharply peaked around e^* . For exponents above the mode, $P(e)$ decays as $P(e) \simeq C 2^{-\alpha e}$ (with $C > 0$), while for exponents below the mode it quickly tends to 0. Consequently, almost all mass is contained in a narrow interval near e^* , and the subnormal bin $e = 0$ carries negligible probability for any non-pathological γ .

The detailed proof is provided in the Appendix A.4. To further validate the theorem, Figure 2 illustrates the exponent probability mass function $P(e)$. The exponent distribution is highly concentrated, which aligns with the empirical results in

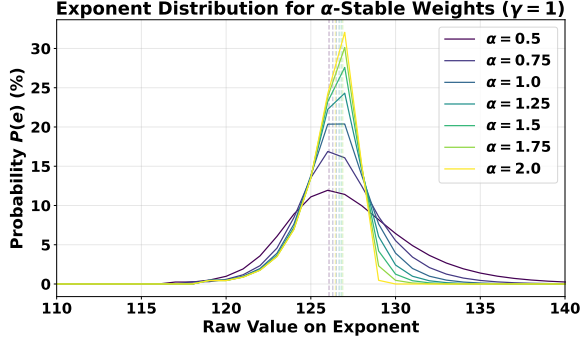


Figure 2: Exponent probability with various α . The vertical line shows the maximizer e^* .

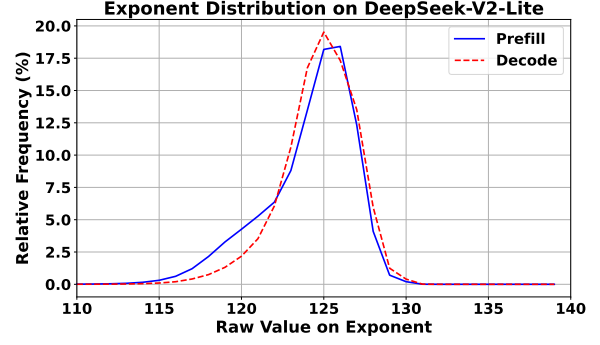


Figure 3: Exponent distribution on the SimpleMath various the prefill (blue) and decode (red) stages.

Figure 1. This theorem further validates the generalization of our observations across various models.

Exponent Distribution Shift Across Different Datasets.

We further examine the shift of exponent distribution in the KV cache across datasets. Figure 1 presents the exponent distribution on four datasets: 2wikimqa, gov_report, musique, multi_news (from LongBench (Bai et al., 2023)), and C4. The concentrated ranges across datasets largely overlap, with only minor shifts. This overlap suggests that a single short code length can be predetermined using a calibration dataset and applied universally, without recomputation per dataset. In other words, a short code length optimized on a representative calibration dataset can be reused, improving encoding efficiency by eliminating the need to repeatedly identify the most frequent exponent range for new data. Results for additional models and datasets are shown in the *Exponent Distribution Shift Across Inference Stages* paragraph of Appendix A.6.

Exponent Distribution Shift Across Inference Stages.

During LLM inference, the size of the KV cache grows with sequence length. A practical consideration when selecting a fixed compression bitwidth is whether the optimal encoding requires adjustment as inference progresses. To investigate this, we examine the exponent distribution across both the prefill and decode stages on the SimpleMath dataset. As shown in Figure 3, the concentrated ranges differ slightly across the two stages, which affects the optimal compression parameters. To maintain a high compression ratio, we therefore design an adjustment mechanism that monitors the KV cache distribution and updates the compression parameters as needed. Results for additional models and datasets are shown in the *Ex-*

ponent Distribution Shift Across Different Datasets paragraph of Appendix A.6.

4 Adaptive Coding for KV cache

Based on the preceding empirical observations, we introduce AdaCode, a bit-exact lossless KV cache compression framework to resolve two key challenges: *selecting an effective code length that maximizes compression*, and *making such coding practical for inference-time KV caches*. We first present how the optimal coding parameters are determined, and then describe the encoding pipeline and storage format used in our system.

4.1 Code Length Selection

The concentration of exponent values motivates a compressed representation where exponents within a high-frequency range are encoded with a short code of L bits, while infrequent exponents outside this range are stored as full-precision outliers. The choice of L presents a trade-off: a larger L increases the short code storage overhead but reduces the number of outliers; a smaller L decreases the short code overhead but increases the outlier count. Therefore, an optimal L exists that minimizes the total storage cost.

We formulate this as an optimization problem. Let W denote all exponent values of KV cache values. We define the L -bit short code to represent the 2^L most frequent exponent values in W . Let M be the count of exponents falling within this frequent set, and N be the count of outliers, i.e., $N = |W| - M$. Storing an outlier requires a full E -bit representation (e.g., $E = 8$ for the BFloat16 values). The total storage cost in bits for all KV cache values is $LM + EN$. However, GPU architectures are designed for massively parallel computation, organizing thousands of threads into

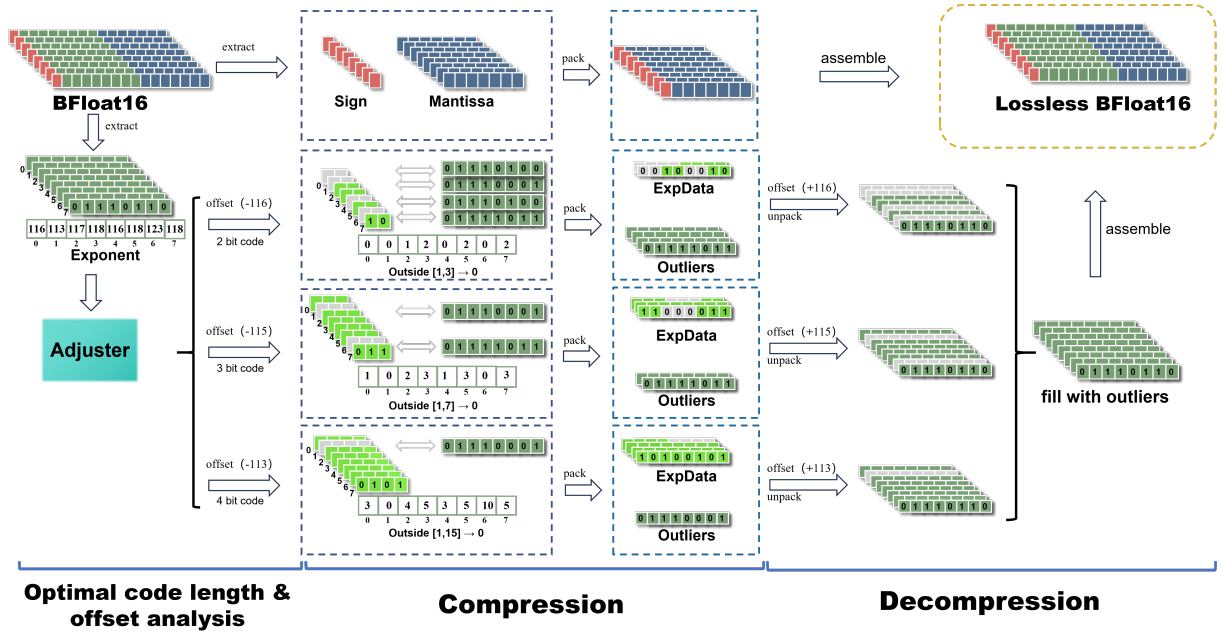


Figure 4: Overview of the AdaCode framework. The input is a block of 8 BFloat16 values, each containing 1 sign bit, 8 exponent bits, and 7 mantissa bits. The framework supports three compressors with different short code lengths (2-bit, 3-bit, and 4-bit). The adjuster module computes the optimal compression parameters (code length and offset) based on the observed exponent distribution and accordingly selects the best compressor. During compression, the sign and mantissa bits of each value are packed into a single uint8, while the exponents are sequentially packed into uint8 units using the selected short code length. During decompression, the packed exponents are unpacked, the offset is added, and outliers are handled to recover the exponents. The sign and mantissa are then reassembled to reconstruct the lossless BFloat16 values.

blocks that execute on streaming multiprocessors (SMs). To leverage the fast access characteristics of SRAM and enable efficient on-the-fly compression and decompression, all exponents (both frequent and outlier) must be stored in a regular, accessible layout. Consequently, in practice, every element is assigned an L -bit codes in the base storage, while outliers additionally require the full E -bit representation. Thus, the actual cost becomes: $\text{Cost}(W) = L|W| + EN$. The optimal short code length L is obtained by minimizing this cost:

$$L = \arg \min_L L|W| + EN, \quad (5)$$

where $M = |\text{Top}_{2^L-1}(\text{Freq}(W))|$, $N = |W| - M$, $\text{Freq}(W)$ returns the frequency of each distinct exponent value in W , $|\text{Top}_{2^L-1}(W)|$ denotes the number of exponents covered by the $2^L - 1$ most frequent values in W . The solution to Equation (5) can be obtained efficiently by searching over the feasible range of L .

We further note that due to the high concentration of the exponent distribution, the $2^L - 1$ most frequent exponents typically form a contiguous interval in the value space as shown in Figure 1 and

2. This property eliminates the need for an explicit lookup table to map between original exponent values and their short codes. Instead, only two parameters are required: a starting offset (the smallest exponent in the frequent set) and the optimal short code length L . All outliers are explicitly assigned a zero marker in the compression representation. Consequently, any exponent e within the contiguous frequent interval $[\text{offset}, \text{offset} + 2^L - 1]$ is compressed as its relative position $e - \text{offset}$, while the detection of an outlier is reduced to checking if the compressed value corresponds to the reserved zero marker.

Adjuster Design. Because the exponent distribution may shift during inference (Figure 3), we introduce a lightweight adjuster module to periodically recompute the optimal short code length and offset. This balances compression effectiveness and inference speed. The adjuster is triggered at a fixed adjustment interval of N generation steps. When activated, it identifies the optimal short code length L and the offset by solving Equation (5). To avoid interference with GPU execution, the adjuster runs asynchronously on the CPU. This design minimizes GPU resource contention and maintains inference

efficiency while ensuring the compression parameters remain near-optimal throughout the generation process.

4.2 Inference-Time Adaptation

Initializing Code Length and Offset. Because the concentrated ranges of the exponent distribution overlap across most of the value range for diverse datasets (Figure 1), we can precompute the optimal short code length L and the corresponding offset using a calibration dataset (*e.g.*, C4) independently for each layer prior to inference. This static initialization eliminates the need to solve Equation (5) at runtime for unseen data, thereby avoiding online search overhead and improving inference efficiency. Furthermore, as shown in Figure 14, this offline calibration does not reduce compression effectiveness and achieves near-optimal memory savings across the evaluated benchmarks.

GPU-friendly Compression Layout. GPUs are efficient at processing regular data layouts. Therefore, the storage layout of compressed data should also be regular. We describe this layout as follows. A BFloat16 value consists of 1 sign bit, 8 exponent bits, and 7 mantissa bits. The minimum addressable unit in typical frameworks (*e.g.*, PyTorch) is 8 bits. We pack the 1-bit sign and the 7-bit mantissa of each value into a single uint8 byte. The optimal short-code length, determined per layer via offline calibration, typically takes one of three values: 2, 3, or 4 bits. As illustrated in Figure 4, packing these variable-length codes into byte-aligned storage requires different grouping strategies:

- **2-bit codes:** Four codes fit exactly into one uint8.
- **3-bit codes:** Eight codes require three uint8s. Consequently, some 3-bit codes may span two adjacent bytes.
- **4-bit codes:** Two codes fit exactly into one uint8.

4.3 Implementation

The overall compression pipeline is summarized in Algorithm 1. First, we initialize the compression parameters (the optimal code length L and the corresponding offset) by solving Equation (5) on the C4 calibration dataset. Those parameters are then used to compress the KV cache throughout inference. Concurrently, the lightweight adjuster periodically recomputes the optimal compression parameters via the same equation every N generation steps (the adjustment interval). If the newly computed parameters differ from the current ones, the adjuster triggers a recompression of the KV

cache to maintain optimality. By combining offline initialization with online adaptation to distribution shifts, our approach achieves a low compression ratio (*i.e.*, small memory footprint) with minimal runtime overhead.

5 Experiments

We implement AdaCode in Triton and integrate it into HuggingFace Transformers (Wolf et al., 2020). Compression parameters are obtained by solving Eq. (5) with default adjustment interval 128. Datasets, models, and other settings are detailed in Sections A.1, A.2, and A.3, respectively.

5.1 Main Results

We evaluate the memory usage of the original KV cache (BFloat16), our compressed KV cache, and the corresponding compression ratio (compressed memory / original memory) across the four LLMs. Table 1 reports per-task compression ratios for each evaluated model. The results demonstrate consistent performance across diverse architectures. The overall compression ratio across all language model tasks ranges from 70.76% to 74.82%, a span of about 4.06 percentage points. Specifically, Qwen3-4B achieves an average ratio of 71.83% (range: 71.37%–73.52%). Mistral-7B-Instruct-v0.3 attains an average of 71.22% (70.76%–72.75%), with musique being its most compressible task. The two MoE models show similarly stable results: Qwen1.5-MoE-A2.7B-Chat averages 71.85% (70.96%–74.82%), while Huihui-MoE-1.2B-A0.6B yields the highest average at 72.21% (71.77%–73.91%), peaking at 73.91% on AIME25. These findings confirm that the effectiveness of our compression method is robust to variations in model scale and architecture.

5.2 Experimental Analysis

Accuracy Comparison with Quantization Baselines. Table 2 compares the accuracy of our method against two KV cache quantization baselines, KIVI (Liu et al., 2024c) and HQQ (Badri and Shaji, 2023). Quantization inherently causes information loss in the input. To verify this effect, we design a copy task where the model is prompted with “Copy the following text exactly as it is” on 100 randomly generated strings. Our method consistently outperforms the quantization baselines across all models, indicating that quantization indeed damages the input representation. We further evaluate this on reasoning tasks that depend

	Qwen3-4B			Mistral-7B-Instruct-v0.3			Qwen1.5-MoE-A2.7B-Chat			Huihui-MoE-1.2B-A0.6B		
Dataset	Orig. (GB)	Comp. (GB)	Ratio (%)	Orig. (GB)	Comp. (GB)	Ratio (%)	Orig. (GB)	Comp. (GB)	Ratio (%)	Orig. (GB)	Comp. (GB)	Ratio (%)
2W.Q.	1.01 $\pm$ 0.06	0.72 $\pm$ 0.04	71.54	0.99 $\pm$ 0.06	0.70 $\pm$ 0.04	70.91	1.35 $\pm$ 0.08	0.96 $\pm$ 0.06	71.14	0.79 $\pm$ 0.05	0.57 $\pm$ 0.03	71.91
G.R.	1.55 $\pm$ 0.09	1.10 $\pm$ 0.06	71.44	1.49 $\pm$ 0.08	1.05 $\pm$ 0.06	70.85	2.06 $\pm$ 0.11	1.46 $\pm$ 0.08	71.02	1.20 $\pm$ 0.07	0.86 $\pm$ 0.05	71.84
M.N.	0.40 $\pm$ 0.04	0.29 $\pm$ 0.03	71.87	0.40 $\pm$ 0.04	0.29 $\pm$ 0.03	71.33	0.54 $\pm$ 0.05	0.38 $\pm$ 0.04	71.48	0.31 $\pm$ 0.03	0.23 $\pm$ 0.02	72.23
musique	2.24 $\pm$ 0.03	1.60 $\pm$ 0.02	71.39	2.21 $\pm$ 0.03	1.56 $\pm$ 0.02	70.76	2.99 $\pm$ 0.04	2.12 $\pm$ 0.02	70.96	1.75 $\pm$ 0.02	1.25 $\pm$ 0.01	71.78
M.E.	0.97 $\pm$ 0.06	0.69 $\pm$ 0.04	71.51	0.95 $\pm$ 0.06	0.67 $\pm$ 0.04	70.89	1.29 $\pm$ 0.08	0.92 $\pm$ 0.06	71.11	0.75 $\pm$ 0.05	0.54 $\pm$ 0.03	71.90
N.Q.	2.42 $\pm$ 0.08	1.73 $\pm$ 0.06	71.42	2.20 $\pm$ 0.06	1.56 $\pm$ 0.04	70.85	3.23 $\pm$ 0.10	2.29 $\pm$ 0.07	71.03	1.88 $\pm$ 0.06	1.35 $\pm$ 0.04	71.82
P.C.	2.12 $\pm$ 0.07	1.51 $\pm$ 0.05	71.40	2.02 $\pm$ 0.06	1.43 $\pm$ 0.05	70.84	2.83 $\pm$ 0.10	2.01 $\pm$ 0.07	71.00	1.65 $\pm$ 0.06	1.18 $\pm$ 0.04	71.82
P.R.E.	1.78 $\pm$ 0.02	1.27 $\pm$ 0.01	71.38	1.73 $\pm$ 0.02	1.23 $\pm$ 0.01	70.79	2.37 $\pm$ 0.03	1.68 $\pm$ 0.02	70.98	1.38 $\pm$ 0.02	0.99 $\pm$ 0.01	71.79
qasper	0.80 $\pm$ 0.06	0.57 $\pm$ 0.05	71.71	0.77 $\pm$ 0.06	0.55 $\pm$ 0.04	71.01	1.07 $\pm$ 0.09	0.76 $\pm$ 0.06	71.25	0.62 $\pm$ 0.05	0.45 $\pm$ 0.04	72.09
qmsum	1.78 $\pm$ 0.07	1.27 $\pm$ 0.05	71.37	1.79 $\pm$ 0.07	1.27 $\pm$ 0.05	70.83	2.37 $\pm$ 0.10	1.68 $\pm$ 0.07	70.99	1.38 $\pm$ 0.06	0.99 $\pm$ 0.04	71.77
H.P.Q.	1.89 $\pm$ 0.07	1.35 $\pm$ 0.05	71.41	1.88 $\pm$ 0.07	1.33 $\pm$ 0.05	70.80	2.53 $\pm$ 0.09	1.79 $\pm$ 0.06	70.99	1.47 $\pm$ 0.05	1.06 $\pm$ 0.04	71.80
lcc	0.46 $\pm$ 0.05	0.33 $\pm$ 0.03	71.82	0.54 $\pm$ 0.05	0.38 $\pm$ 0.04	71.14	0.61 $\pm$ 0.06	0.44 $\pm$ 0.04	71.42	0.36 $\pm$ 0.04	0.26 $\pm$ 0.03	72.16
R.P.	1.48 $\pm$ 0.10	1.06 $\pm$ 0.07	71.45	1.64 $\pm$ 0.10	1.16 $\pm$ 0.07	70.86	1.98 $\pm$ 0.13	1.40 $\pm$ 0.09	71.01	1.15 $\pm$ 0.08	0.83 $\pm$ 0.05	71.81
samsum	1.29 $\pm$ 0.08	0.92 $\pm$ 0.06	71.39	1.35 $\pm$ 0.09	0.96 $\pm$ 0.06	70.85	1.72 $\pm$ 0.11	1.22 $\pm$ 0.08	71.04	1.00 $\pm$ 0.06	0.72 $\pm$ 0.05	71.81
trec	0.86 $\pm$ 0.05	0.62 $\pm$ 0.03	71.54	0.85 $\pm$ 0.05	0.60 $\pm$ 0.03	70.94	1.15 $\pm$ 0.06	0.82 $\pm$ 0.04	71.15	0.67 $\pm$ 0.04	0.48 $\pm$ 0.03	71.89
triviaqa	1.55 $\pm$ 0.10	1.11 $\pm$ 0.07	71.42	1.52 $\pm$ 0.09	1.08 $\pm$ 0.06	70.82	2.07 $\pm$ 0.13	1.47 $\pm$ 0.09	71.01	1.21 $\pm$ 0.08	0.87 $\pm$ 0.05	71.80
LCB	0.22 $\pm$ 0.01	0.16 $\pm$ 0.01	72.49	0.21 $\pm$ 0.01	0.15 $\pm$ 0.01	71.73	0.29 $\pm$ 0.01	0.21 $\pm$ 0.01	72.09	0.17 $\pm$ 0.01	0.12 $\pm$ 0.00	72.81
mbpp	0.14 $\pm$ 0.00	0.10 $\pm$ 0.00	71.92	0.13 $\pm$ 0.00	0.09 $\pm$ 0.00	71.35	0.19 $\pm$ 0.00	0.14 $\pm$ 0.00	73.64	0.11 $\pm$ 0.00	0.08 $\pm$ 0.00	72.33
AIME24	0.15 $\pm$ 0.00	0.11 $\pm$ 0.00	73.29	0.14 $\pm$ 0.00	0.10 $\pm$ 0.00	72.75	0.20 $\pm$ 0.00	0.15 $\pm$ 0.00	74.55	0.12 $\pm$ 0.00	0.09 $\pm$ 0.00	73.68
AIME25	0.16 $\pm$ 0.00	0.11 $\pm$ 0.00	<u>73.52</u>	0.14 $\pm$ 0.00	0.10 $\pm$ 0.00	72.65	0.21 $\pm$ 0.00	0.15 $\pm$ 0.00	74.68	0.12 $\pm$ 0.00	0.09 $\pm$ 0.00	<u>73.91</u>
R.B.	0.15 $\pm$ 0.00	0.11 $\pm$ 0.00	73.33	0.13 $\pm$ 0.00	0.10 $\pm$ 0.00	<u>72.75</u>	0.20 $\pm$ 0.00	0.15 $\pm$ 0.00	<u>74.82</u>	0.12 $\pm$ 0.00	0.09 $\pm$ 0.00	73.70
S.M.	0.14 $\pm$ 0.00	0.10 $\pm$ 0.00	71.68	0.13 $\pm$ 0.00	0.09 $\pm$ 0.00	71.12	0.19 $\pm$ 0.00	0.14 $\pm$ 0.00	73.23	0.11 $\pm$ 0.00	0.08 $\pm$ 0.00	72.07
Avg.	1.07 $\pm$ 0.05	0.77 $\pm$ 0.03	71.83	1.05 $\pm$ 0.04	0.75 $\pm$ 0.03	71.22	1.43 $\pm$ 0.06	1.02 $\pm$ 0.04	71.85	0.83 $\pm$ 0.04	0.60 $\pm$ 0.03	72.21

Table 1: Memory usage comparison across four large language models, evaluating original KV cache memory (Orig.) and compressed KV cache memory (Comp.). Values are presented with the standard error shown in the lower right corner. The compression ratio is calculated as $\text{Comp./Orig.} \times 100\%$; a lower ratio indicates higher compression. Boldface denotes the lowest compression ratio per model; underline indicates the highest compression ratio.

heavily on input sequences: RiddleBench and SimpleMath. The results show that the quantization baselines indeed underperform on these tasks. In contrast, our method achieves stable and superior accuracy, producing results identical to the uncompressed BFloat16 baseline and surpassing typical KV cache quantization methods. The performance indicates that our method faithfully preserves the original model’s behavior, and is particularly effective for tasks that critically depend on in-context information.

Throughput-per-memory Comparison with Lossless Method. We use throughput-per-memory (tokens/s/GB) to normalize across methods with different compression ratios (Mamo et al., 2026); higher values indicate better efficiency. This metric captures the trade-off between memory savings and inference speed, enabling a fair comparison between lossless methods that achieve different memory footprints. As shown in Figure 5, our method achieves the highest throughput per memory among all tested baselines (original uncompressed, CPU offloading, and Neuzip) across every model. For example, on Mistral-7B, our method reaches 2762

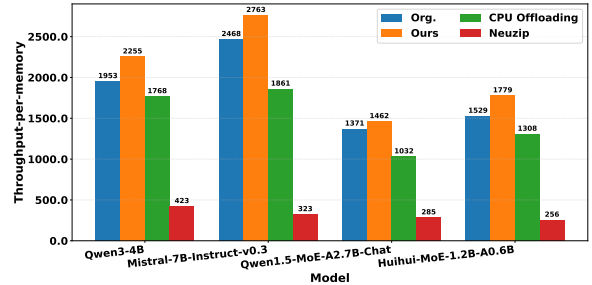


Figure 5: Memory efficiency comparison on multi_news dataset.

tokens/s/GB, outperforming the original by about 11%. These results demonstrate that our adaptive coding strategy not only reduces memory usage effectively but also maintains low runtime overhead.

Calibration Initialization Impact on TTFT. To quantify the overhead of computing compression parameters on-the-fly, we compare the Time-To-First-Token (TTFT) under two settings: (1) using precomputed compression parameters from a calibration dataset (C4), and (2) performing real-time computation of the short code length and offset for each KV cache block during the prefill stage.

Tasks	Qwen3-4B			Mistral-7B-Instruct-v0.3			Qwen1.5-MoE-A2.7B-Chat			Huihui-MoE-1.2B-A0.6B		
	KIVI	HQQ	Ours	KIVI	HQQ	Ours	KIVI	HQQ	Ours	KIVI	HQQ	Ours
Copy	57.00	62.00	77.00	57.00	53.00	58.00	22.00	13.00	24.00	0.00	19.00	80.00
RiddleBench	11.69	12.55	13.41	5.24	5.24	5.30	6.04	6.68	6.74	1.61	2.70	3.97
SimpleMath	38.70	42.50	47.30	43.40	42.40	45.70	4.60	4.50	4.70	0.70	0.90	1.90
Avg.	35.80	39.02	45.90	35.21	33.55	36.33	10.88	8.06	11.81	0.77	7.53	28.62

Table 2: Accuracy (%) comparison with KV cache quantization baselines. Our method (Ours) is lossless and yields identical results to uncompressed BFloat16.

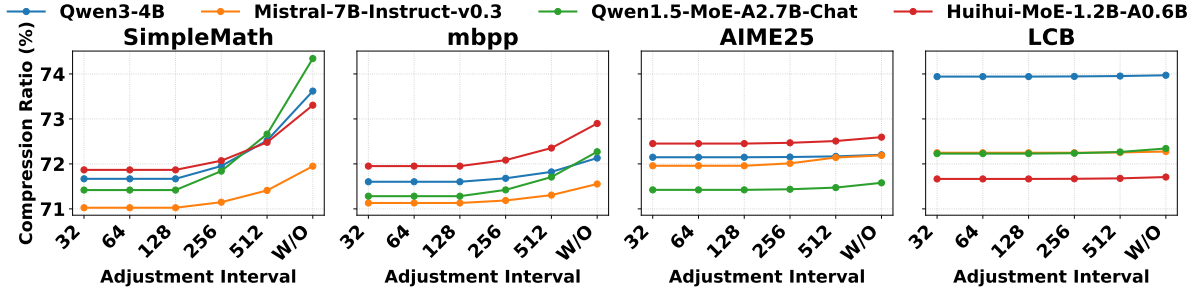


Figure 6: Compression ratio for different adjustment intervals across four LLMs. ‘W/O’ denotes without the adjuster.

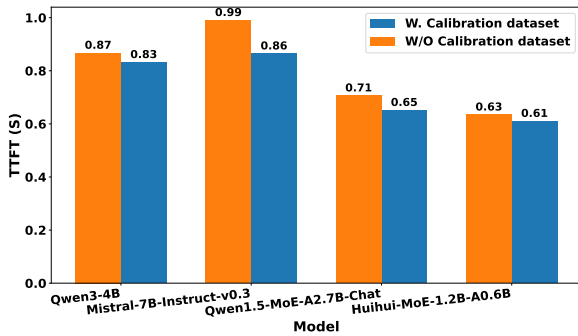


Figure 7: Impact of calibration dataset on TTFT on multi_news dataset.

As shown in Figure 7, initialization with the calibration dataset yields consistent TTFT reductions across all models, with relative improvements of 4.21%, 14.56%, 8.41%, and 3.96% across four LLMs. These results confirm that offline calibration effectively eliminates runtime optimization overhead during the prefill phase.

Adjustment Interval Impact on Compression Ratio. To evaluate the impact of the adjuster’s update frequency, we measure the compression ratio across four LLMs for different adjustment intervals (number of generation steps between compression parameter updates). Because the adjuster operates under shifting distributions, we only visualize the samples where the distribution difference between using and not using the adjuster is

largest. As shown in Figure 6, the compression ratio decreases as the adjustment interval becomes smaller. Without the adjuster, the compression ratio is highest, leading to increased GPU memory usage. When the adjustment interval reaches 128, the compression ratio converges and does not improve with further decreases in interval. For datasets with large distribution shifts during generation (e.g., SimpleMath and MBPP), the adjuster plays a key role in reducing the compression ratio, achieving up to a 3% reduction on the SimpleMath dataset for the Qwen1.5-MoE-A2.7B-Chat model. For datasets with small distribution shifts (e.g., AIME25 and LCB), the reduction in compression ratio is modest.

6 Conclusion

In this work, we propose adaptive coding for lossless KV cache compression motivated by the high concentration and stability of BFloat16 exponent distributions. By encoding frequent exponents with a adaptive short codes, coupled with offline calibration and a lightweight online adjuster, our method achieves an average compression ratio of 71.78% without accuracy loss. This effectively reduces memory pressure, maintains competitive inference throughput, offering a practical solution for scaling context length.

Limitations

Our work focuses primarily on the BFloat16 format, as it is the prevailing representation for inference in contemporary LLMs. Compression strategies for other numerical formats (e.g., FP16, FP8, or FP32) may differ and are not explored here. Furthermore, our implementation and evaluation are centered on GPU platforms; performance on other hardware such as CPUs, TPUs, or custom accelerators may require additional platform-specific optimizations and remains to be studied.

ETHICS STATEMENT

This work focuses on the technical problem of lossless KV cache compression and does not involve human subjects, personal data, or sensitive information. All datasets used are publicly available and widely adopted in the research community. We foresee no specific ethical risks beyond those generally associated with deploying more efficient large language models.

Acknowledgments

Large language models were employed only for grammatical checking and minor language polishing of the manuscript; they were not used in designing experiments, generating results, or forming scientific conclusions.

This work was supported by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant Nos. 12125103, U24A2002 and 62506077, and the Natural Science Foundation of Hubei Province under Grant No. 2025AFA002.

References

- Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *CoRR*, abs/2108.07732.
- Hicham Badri and Appu Shaji. 2023. [Half-quadratic quantization of large machine learning models](#).
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. [Longbench: A bilingual, multitask benchmark for long context understanding](#). *Preprint*, arXiv:2308.14508.
- Zhihang Cai, Xingjun Zhang, Zhendong Tan, and Zheng Wei. 2025. [Nqkv: A kv cache quantization scheme based on normal distribution characteristics](#). *Preprint*, arXiv:2505.16210.
- Haoran Guan, Junhao Lv, Xi Chen, and Lei Qi. 2026. [Lightweighting large models: a review of model compression techniques](#). *Data Intelligence*, 8(1):92–136.
- Deepon Halder, Alan Saji, Thanmay Jayakumar, Anoop Kunchukuttan, Ratish Puduppully, and Raj Dabre. 2026. [RiddleBench: A new generative reasoning benchmark for LLMs](#). In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4363–4372, Rabat, Morocco. Association for Computational Linguistics.
- Song Han, Huizi Mao, and William J. Dally. 2015. [Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding](#). *arXiv: Computer Vision and Pattern Recognition*.
- Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. [Neuzip: Memory-efficient training and inference with dynamic compression of neural networks](#). *ArXiv*, abs/2410.20650.
- Moshik Hershcovitch, Andrew Wood, Leshem Choshen, Guy Gironmsky, Roy Leibovitz, Ilias Ennmouri, Michal Malka, Peter Chin, Swaminathan Sundararaman, and Danny Harnik. 2024. [Zipnn: Lossless compression for AI models](#). *CoRR*, abs/2411.05239.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. [Kvquant: Towards 10 million context length LLM inference with KV cache quantization](#). In *NeurIPS*.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. [Livecodebench: Holistic and contamination free evaluation of large language models for code](#). *arXiv preprint arXiv:2403.07974*.
- Prannay Kaul, Chengcheng Ma, Ismail Elezi, and Jiankang Deng. 2025. [From attention to activation: Unraveling the enigmas of large language models](#). In *International Conference on Learning Representations*, volume 2025, pages 34531–34582.
- Keith Knight. 1997. [Stable non-gaussian random processes](#) gennady samorodnitsky and murad s. taqqu chapman and hall, 1994. *Econometric Theory*, 13(1):133–142.
- Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2025a. [A survey on large language model acceleration based on KV cache management](#). *Trans. Mach. Learn. Res.*, 2025.
- Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. 2025b. [CommVQ: Commutative vector quantization for KV cache compression](#).

- In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 36831–36845. PMLR.
- Xin Lin, Yajiao Wang, Zhixiong Zhang, and Mengting Zhang. 2025. Scientific claim recognition via staged fine-tuning with lora. *Data Intelligence*, 7(2):303–335.
- Peiyu Liu, Ze-Feng Gao, Xin Zhao, Yipeng Ma, Tao Wang, and Ji-Rong Wen. 2024a. Unlocking data-free low-bit quantization with matrix decomposition for KV cache compression. In *ACL (1)*, pages 2430–2440. Association for Computational Linguistics.
- Peiyu Liu, Zikang Liu, Ze-Feng Gao, Dawei Gao, Wayne Xin Zhao, Yaliang Li, Bolin Ding, and Ji-Rong Wen. 2024b. Do emergent abilities exist in quantized large language models: An empirical study. In *LREC/COLING*, pages 5174–5190. ELRA and ICCL.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. **KIVI: A tuning-free asymmetric 2bit quantization for KV cache**. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344. PMLR.
- Oteo Mamo, Olga Kogiou, Hyunjin Yi, and Weikuan Yu. 2026. **Comparative characterization of kv cache management strategies for llm inference**. *Preprint*, arXiv:2604.05012.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. **Exploring the limits of transfer learning with a unified text-to-text transformer**. *arXiv e-prints*.
- Akshat Sharma, Hangliang Ding, Jianping Li, Neel Dani, and Minjia Zhang. 2024. Minikv: Pushing the limits of LLM inference via 2-bit layer-discriminative KV cache. *CoRR*, abs/2411.18077.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. **Transformers: State-of-the-art natural language processing**. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, and 3 others. 2024. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of CVPR*.
- Tianyi Zhang, Mohsen Hariri, Shaochen (Henry) Zhong, Vipin Chaudhary, Yang Sui, Xia Hu, and Anshumali Shrivastava. 2025. **70% size, 100% accuracy: Lossless llm compression for efficient gpu inference via dynamic-length float (dfloat11)**. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 98966–98994. Curran Associates, Inc.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, and 4 others. 2026. **A survey of large language models**. *Frontiers of Computer Science*, 20(12).

A Appendix

A.1 Data Description

We use C4 (Raffel et al., 2019) as the calibration dataset to precompute the offset and short-code length for each model. Compression efficiency is evaluated on a comprehensive set of benchmarks. For language models, we use LongBench (Bai et al., 2023), including the 16 different tasks. We additionally evaluate on six established benchmarks: livecodebench (LCB) (Jain et al., 2024), mbpp (Austin et al., 2021), AIME24¹, AIME25², RiddleBench (Halder et al., 2026) and SimpleMath³. For the vision-language model, we evaluate on MMMU (Yue et al., 2024), a benchmark spanning 30 academic subjects.

We conduct experiments using publicly available benchmarks to ensure reproducibility and fair comparison. Below we describe each dataset, including its purpose, size, and the specific splits used in our study.

C4 (Raffel et al., 2019) is a large-scale, English-language text corpus constructed from filtered Common Crawl web data and made available under the Open Database Commons Attribution License (ODC-BY). We employ its substantial training split (approximately 365 million samples) as a representative and diverse calibration dataset. This allows us to robustly precompute the stable compression parameters (offsets and short-code lengths) for each LLM prior to task-specific inference.

LongBench (Bai et al., 2023) is a comprehensive bilingual benchmark under the Apache License 2.0,

¹<https://huggingface.co/datasets/math-ai/aime24>

²<https://huggingface.co/datasets/math-ai/aime25>

³<https://huggingface.co/datasets/ProCreations/SimpleMath>

designed for multi-task evaluation of long-context understanding. To thoroughly assess the generality of our method, we utilize its 16 constituent sub-tasks, which span a wide spectrum of capabilities:

- **Question Answering:** 2wikimqa (2W.Q., 200), hotpotqa (H.P.Q., 200), multifielddqa_en (M.E., 150), musique (200), narrativeqa (N.Q., 200), qasper (200), triviaqa (200).
- **Summarization:** gov_report (G.R., 200), multi_news (M.N., 200), qmsum (200), samsum (200).
- **Code Completion:** lcc (500), repobench-p (R.P., 500).
- **Retrieval and Classification:** passage_count (200), passage_retrieval_en (P.R.E., 200), trec (200).

In total, this amounts to evaluation on 3,750 test examples, using the official test splits to ensure standardized and comparable results.

LiveCodeBench (LCB) (Jain et al., 2024) is a continuously updated benchmark for evaluating code generation and understanding capabilities of large language models, released under the Creative Commons (CC) license. We use a recent release (spanning problems from May 2023 to January 2025), which contains 880 diverse programming problems, to test our method’s robustness on evolving, real-world coding tasks.

MBPP (Austin et al., 2021) is a benchmark comprising approximately 1,000 crowd-sourced entry-level Python programming problems, released under the CC BY 4.0 license. It is designed to evaluate fundamental coding skills. We evaluate on its standard test split of 974 examples.

AIME24 and **AIME25** contain challenging problems from the respective American Invitational Mathematics Examinations and are distributed under the Apache License 2.0. We evaluate on their test splits (30 problems each) to assess our method’s performance on complex, multi-step mathematical reasoning.

SimpleMath (S.M.) contains 100,000 basic arithmetic problems under the MIT License. We use its numeric problems to evaluate performance against quantization-based methods.

RiddleBench (R.B.) (Halder et al., 2026) consists of 1,737 challenging English puzzles designed to

assess reasoning capabilities of large language models. Released under the CC0 license, it is used to compare reasoning performance with quantization-based methods.

MMMU (Yue et al., 2024) is a comprehensive multi-disciplinary, multi-modal understanding benchmark covering 30 academic subjects. We use its validation split (30 examples per subject) to calibrate compression parameters specifically for the vision-language model. The benchmark is made available under the Apache License 2.0. For the final performance evaluation, we utilize the full test set of 10,500 examples, distributed across subjects as follows:

- **Science and Engineering:** Chemistry (603), Biology (345), Physics (408), Math (505), Computer_Science (371), Materials (458), Mechanical_Engineering (M.E., 429), Electronics (256), Energy_and_Power (432).
- **Medical and Health Sciences:** Pharmacy (430), Clinical_Medicine (325), Basic_Medical_Science (B.M.S., 326), Public_Health (509), Diagnostics_and_Laboratory_Medicine (D.L.M., 162).
- **Humanities and Social Sciences:** History (278), Literature (112), Sociology (252), Psychology (305), Geography (565), Art_Theory (429).
- **Business and Economics:** Accounting (380), Finance (355), Economics (267), Marketing (181).
- **Arts and Design:** Art (231), Design (169), Music (334).
- **Applied Sciences:** Agriculture (A.E., 287), Architecture_and_Engineering (551), Manage (245).

Our use of all aforementioned datasets is strictly for research purposes, specifically for the calibration and evaluation of language model compression techniques. This usage aligns with the intended purposes as defined by their creators (e.g., model pre-training, benchmarking, and evaluation) and is permitted under their respective open licenses (ODC-BY, Apache 2.0, CC BY 4.0, etc.). Any derived data (e.g., precomputed compression parameters) generated from these datasets in this study

are intended for research use only and will be distributed under terms compatible with the original licenses, ensuring they are not used outside of research contexts.

A.2 Model Description

We evaluate five representative models to assess the generality of our method: two dense models (Qwen3-4B ⁴ and Mistral-7B-Instruct-v0.3 ⁵, abbreviated as Mistral-7B), two mixture-of-experts (MoE) models (Qwen1.5-MoE-A2.7B-Chat ⁶ (Qwen1.5-MoE) and Huihui-MoE-1.2B-A0.6B ⁷ (Huihui-MoE)), one vision-language model (Qwen3-VL-8B-Instruct ⁸) and one MLA model (DeepSeek-V2-Lite ⁹).

Qwen3-4B is a decoder-only large language model representing the latest generation of the Qwen series. It contains 4.0 billion parameters across 36 transformer layers and natively supports a context length of 32,768 tokens. The model is publicly released under the Apache License 2.0.

Mistral-7B-Instruct-v0.3 is an instruction-tuned variant of the Mistral-7B-v0.3 model. It extends the vocabulary size to 32,768 and consists of 32 transformer layers. The model is distributed under the Apache License 2.0.

Qwen1.5-MoE-A2.7B-Chat is a Mixture-of-Experts (MoE) decoder-only language model with a total of 14.3 billion parameters, of which approximately 2.7 billion are activated during inference. It comprises 24 transformer layers and supports a context length of 32,768 tokens. The model is released under the Tongyi Qianwen license.

Huihui-MoE-1.2B-A0.6B is a MoE language model developed by Huihui.ai, featuring 1.2 billion total parameters with 0.6 billion activated during forward passes. The model contains 28 transformer layers and supports a context length of 40,960 tokens. It is released under the Apache License 2.0.

Qwen3-VL-8B-Instruct is a vision-language model that integrates a visual encoder with a 36-layer text decoder. The visual encoder processes images with a patch size of 16 and outputs visual

features projected into the text embedding space. It is distributed under the Apache License 2.0.

DeepSeek-V2-Lite is a decoder-only MoE model with 16B total parameters (2.4B activated per token). Its Multi-head Latent Attention (MLA) compresses KV cache for efficient inference. The model has 27 layers and a 32k context length. Code is under MIT, while the model uses the DeepSeek Model License (permitted for research use).

All models listed above are employed in this research under the terms of their respective open-source licenses. Our usage—loading model weights for inference, calibration, and evaluation—is consistent with the intended research and development purposes for which these models were publicly released. No modifications were made to the core model weights beyond the application of our compression method. Results and any released model adaptations (*e.g.*, compression configurations) based on these artifacts are intended for research use only.

A.3 Experimental Settings

We implement the compression and decompression kernels of AdaCode in Triton and integrate them into the HuggingFace Transformers framework (Wolf et al., 2020). The compression parameters (short code length and offset) are obtained by solving the optimization objective in Eq. (5). The default adjustment interval is 128. We use a batch size of 1 unless otherwise specified. All experiments are conducted on a server with an NVIDIA A800 GPU (80GB), using the default generation configuration provided by HuggingFace. The total runtime for evaluating each model across all datasets averages approximately two days.

A.4 Proofs

Proof for Theorem 3.1. By symmetry, for $x > 0$ we have $F(x) = \frac{1}{2} + \int_0^x f(u) du$, and $f(x) > 0$ for all $x \in \mathbb{R}$.

Existence of a maximiser. Treat the exponent as a continuous variable via $s = 2^{e-127} > 0$ and define $g(s) = 2[F(rs) - F(s)]$, so that $P(e) = g(2^{e-127})$.

Because F is continuously differentiable, g is differentiable on $(0, \infty)$ with

$$g'(s) = 2[r f(rs) - f(s)].$$

Study the limits of g :

⁴<https://huggingface.co/Qwen/Qwen3-4B>

⁵<https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3>

⁶<https://huggingface.co/Qwen/Qwen1.5-MoE-A2.7B-Chat>

⁷<https://huggingface.co/huihui-ai/Huihui-MoE-1.2B-A0.6B>

⁸<https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct>

⁹<https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite>

- As $s \rightarrow 0^+$, $F(rs) - F(s) \rightarrow 0$, hence $g(s) \rightarrow 0$.
- As $s \rightarrow \infty$, $F(x) \rightarrow 1$; hence $F(rs) - F(s) \rightarrow 0$ and $g(s) \rightarrow 0$.

Since $g(s) > 0$ for all $s > 0$ (the interval length $(r-1)s > 0$ and the density is strictly positive), the continuous function g attains a *global maximum* at some interior point $s^* > 0$. At any such maximiser the first-order condition $g'(s^*) = 0$ must hold, which yields

$$rf(rs^*) = f(s^*). \quad (6)$$

Thus there exists at least one maximiser, and the corresponding exponent $e^* = 127 + \log_2 s^*$.

Scale invariance. The scaling property of stable laws gives $X/\gamma \sim S(\alpha, 0, 1, 0)$, so

$$f(x) = \frac{1}{\gamma} f_0\left(\frac{x}{\gamma}\right), \quad F(x) = F_0\left(\frac{x}{\gamma}\right),$$

where f_0 and F_0 are the density and cumulative distribution function of $S(\alpha, 0, 1, 0)$. Insert these expressions into (6):

$$r \frac{1}{\gamma} f_0\left(\frac{rs^*}{\gamma}\right) = \frac{1}{\gamma} f_0\left(\frac{s^*}{\gamma}\right) \implies rf_0(rt^*) = f_0(t^*),$$

where $t^* = \frac{s^*}{\gamma}$. Hence t^* satisfies the scale-free equation $rf_0(rt) = f_0(t)$, which involves neither γ nor the initial scale. Therefore every maximiser fulfills $s^* = \gamma t^*$, and from $e^* = 127 + \log_2 s^*$ we obtain (3). The peak probability follows by substituting the scaled variables:

$$P(e^*) = 2[F(rs^*) - F(s^*)] = 2[F_0(rt^*) - F_0(t^*)],$$

which is manifestly independent of γ and depends only on α and r .

Concentration and tail decay. We now prove that the mass outside a neighborhood of e^* is negligible.

- *Large e (right tail).* For s large, use the known power-law tail of symmetric α -stable distributions (Property 1.2.15 by (Knight, 1997)):

$$\mathbb{P}(|X| \geq x) \sim C_{\alpha,\gamma} x^{-\alpha}, \quad x \rightarrow \infty,$$

where $C_{\alpha,\gamma} > 0$ depends only on α and γ . Rewrite $P(e)$ using the survival function: $P(e) = \mathbb{P}(s \leq |X| < rs) = \mathbb{P}(|X| \geq s) - \mathbb{P}(|X| \geq rs)$. For s sufficiently large (which is certainly the case for exponents near 254, where $s = 2^{127}$ is astronomically larger

than typical γ), the asymptotic approximation applies, giving

$$\begin{aligned} P(e) &\sim C_{\alpha,\gamma} s^{-\alpha} - C_{\alpha,\gamma} (rs)^{-\alpha} \\ &= C_{\alpha,\gamma} s^{-\alpha} (1 - r^{-\alpha}). \end{aligned}$$

Since $s = 2^{e-127}$, we have $s^{-\alpha} \propto 2^{-\alpha e}$, establishing the exponential decay $P(e) \simeq C 2^{-\alpha e}$ for large e .

- *Small e (left tail).* When e is small, s is tiny. A first-order Taylor expansion of F around 0 gives $F(rs) - F(s) = \int_s^{rs} f(u) du \approx f(0)(r-1)s \rightarrow 0$ as $s \rightarrow 0^+$, so $P(e) \rightarrow 0$. In particular, the subnormal bin $e = 0$ corresponds to $s \approx 2^{-126}$, an interval of width $\sim 2^{-126}$. Because $P(0) \approx 2f(0)(r-1)2^{-126}$ is vanishingly small unless γ is exceptionally tiny, its contribution is negligible.

Because the total probability sums to 1, the exponential decay on the right forces the mass to be concentrated: for any $\varepsilon > 0$ we can choose a finite set of exponents $\mathcal{E}_\varepsilon$ (e.g., an interval around e^*) such that $\sum_{e \notin \mathcal{E}_\varepsilon} P(e) < \varepsilon$. Hence the distribution is sharply peaked around e^* , regardless of whether the maximiser is unique. This completes the proof. $\square$

A.5 Adaptive Coding Algorithm

The details of our adaptive coding algorithm are shown in Algorithm 1. It consists of two procedures: `GetCompressionParameters` computes the compression parameters using a calibration dataset, and `AdaptiveCompression` adjusts the parameters during inference to accommodate distribution shifts.

A.6 Experimental Analysis

BFloat16 Distribution of KV Cache. We analyze the distribution of sign, exponent, and mantissa in the KV cache on the C4 dataset. As shown in Figure 11, we observe a pattern similar to that of model weights: the sign and mantissa distributions are relatively flat, whereas the exponent distribution is highly concentrated. This indicates that statistical redundancy is substantially lower in the sign and mantissa bits than in the exponent bits. The concentrated exponent distribution also suggests that our compression method can be applied effectively across a wide range of models.

Algorithm 1 Implementation of Adaptive Coding for Lossless KV Cache Compression

```

1: procedure GETCOMPRESSIONPARAMETERS( $W$ )
2:   Solve  $L^* = \arg \min_L L|W| + EN$   $\triangleright$  Eq. (5)
3:   Obtain the minimum index of  $\text{Top}_{2L^*}(\text{Freq}(W))$  as the offset  $o^*$ 
4:   return  $(L^*, o^*)$ 
5: end procedure
6: procedure ADAPTIVECOMPRESSION( $T, N, \mathcal{D}_{\text{calib}}$ )
7:    $(L_{\text{curr}}, o_{\text{curr}}) \leftarrow \text{GetCompressionParameters}(\mathcal{D}_{\text{calib}})$ 
8:   Initialize compression parameters via  $(L_{\text{curr}}, o_{\text{curr}})$ 
9:   for  $t \leftarrow 1$  to  $T$  do
10:    Generate token  $x_t$ , update KV cache  $\mathcal{C}_o$ 
11:    Compress KV cache to  $\mathcal{C}$  via  $(L_{\text{curr}}, o_{\text{curr}})$ 
12:    if  $t \bmod N = 0$  then  $\triangleright$  Periodic adaptation
13:       $(L_{\text{new}}, o_{\text{new}}) \leftarrow \text{GetCompressionParameters}(\mathcal{C}_o)$ 
14:      if  $(L_{\text{new}}, o_{\text{new}}) \neq (L_{\text{curr}}, o_{\text{curr}})$  then
15:        Recompress  $\mathcal{C}$  using  $(L_{\text{new}}, o_{\text{new}})$ 
16:         $(L_{\text{curr}}, o_{\text{curr}}) \leftarrow (L_{\text{new}}, o_{\text{new}})$ 
17:      end if
18:    end if
19:  end for
20:  return  $\mathcal{C}$ 
21: end procedure

```

Exponent Distribution Shift Across Different Datasets. To further examine KV cache distribution shifts across different datasets, we present results for the remaining models. As shown in Figures 9, 8, and 10, the exponent distribution across different datasets overlaps over most of the range, with only minor shifts. These findings across multiple models further support the use of precomputed compression parameters from a calibration dataset, and suggest that compression parameters can be adjusted adaptively to balance time cost and compression ratio.

Exponent Distribution Shift Across Inference Stages. We further examine the exponent distribution shift across four models and four datasets. As shown in Figure 12, the exponent distributions largely overlap across nearly all models and datasets. Even for the Qwen1.5-MoE-A2.7B-Chat model on the C4 dataset, the concentrated ranges still overlap by more than half. These consistent observations suggest that the compression parameters do not need frequent recomputation.

Optimal Code Length Analysis. Table 3 shows the distribution of optimal short code lengths across all KV cache layers on the C4 dataset. Across the four models, 3-bit codes are the most common, while 2-bit and 4-bit codes appear rarely or not at all.

Outlier Ratios are Low. In our method, the most frequent exponent values are compressed using a short code, while the remaining values are treated

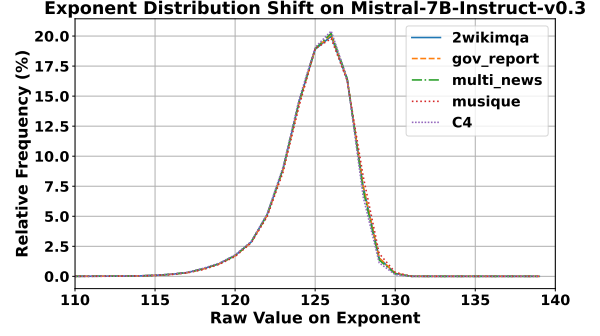


Figure 8: Distribution shift of the Mistral-7B-Instruct-v0.3 model across five different datasets.

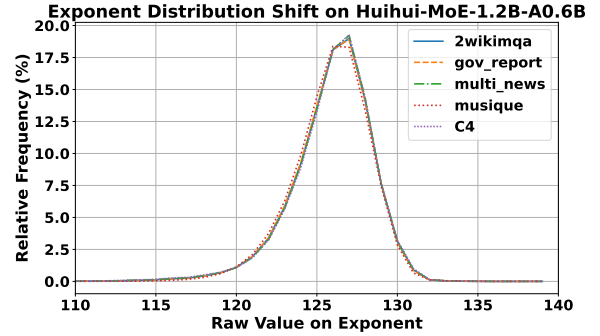


Figure 9: Distribution shift of the Huihui-MoE-1.2B-A0.6B model across five different datasets.

as outliers and stored in their original 8-bit representation. Figure 13 shows the proportion of these outlier values across four different LLMs on LongBench after generation. The outlier ratios range from 3% to 8%, with a median value below 6% for all models. This low proportion of outliers demonstrates that our strategy of leaving them uncompressed incurs minimal storage overhead, thereby validating the statistical redundancy of the KV cache.

Calibration Initialization Impact on Compression Ratio. To evaluate the impact of using a calibration dataset on compression ratio, we measure the compression ratio on the multi_news task across four LLMs under two settings: using pre-computed compression parameters from the C4 calibration dataset, and computing the short code

Short-code length(%)	2-bit	3-bit	4-bit
Qwen3-4B	0.0	100.0	0.0
Mistral-7B-Instruct-v0.3	1.6	98.4	0.0
Huihui-MoE-1.2B-A0.6B	0.0	98.2	1.8
Qwen1.5-MoE-A2.7B-Chat	0.0	100.0	0.0

Table 3: The ratios of optimal short code length across four LLMs.

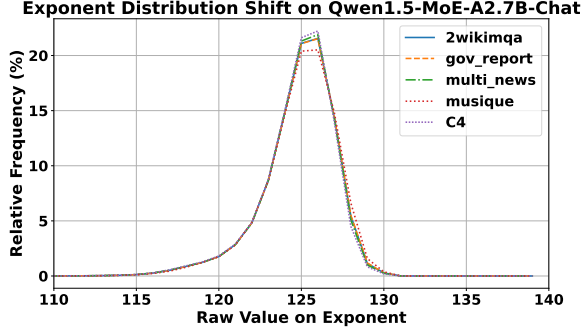


Figure 10: Distribution shift of the Qwen1.5-MoE-A2.7B-Chat model across five different datasets.

length and offset in real time for each cache during the prefill stage. As shown in Figure 14, most models achieve nearly the same compression ratio (difference $<1\%$), and only for the Qwen1.5-MoE-A2.7B-Chat model does the difference remain small (about 2%). These results verify the effectiveness of using calibration datasets to accelerate time-to-first-token (TTFT) without significantly increasing the compression ratio. Additionally, prefilled inputs can be recompressed when the adjuster updates the compression parameters, thereby reducing the compression ratio gap.

Adjustment Intervals Impact on Generation Time. Because the adjuster runs asynchronously on the CPU and the re-compression overhead from decode distribution shifts is small, its impact on generation time is relatively limited. To verify this, we evaluate the time cost of different adjustment intervals (including no adjuster) on several datasets, focusing on the decode stage. We measure generation time for 1024 new tokens with batch size 1 across four datasets. As shown in Figure 15, the time overhead is modest. An interval of 32 steps results in slightly higher cost than other intervals (*e.g.*, LCB and mpp for the Qwen3-4B model). We use 128 as the default interval, as it already provides a converged compression ratio.

Extension to VLM. To evaluate the generalizability of our method beyond text-only models, we apply it to the VLM Qwen3-VL-8B-Instruct using a batch size of 4. We calibrate the compression parameters using the validation split of the MMMU benchmark and report results on its test split. The detailed memory usage and compression ratios are presented in Table 5. Our method achieves consistent compression ratios between 70.21% and 70.31% across all subjects, with an average of 70.26%. These results confirm that

our compression approach effectively extends to vision-language models, maintaining its effectiveness across multimodal inference tasks.

Dataset	MLA	Comp.	Ratio
2wikimqa	1.94 ± 0.11	1.39 ± 0.08	71.63%
G.R.	2.95 ± 0.16	2.11 ± 0.12	71.53%
M.N.	0.80 ± 0.08	0.58 ± 0.06	71.99%
musique	4.31 ± 0.05	3.08 ± 0.04	71.51%
M.E.	1.85 ± 0.12	1.33 ± 0.08	71.63%
narrativeqa	4.57 ± 0.14	3.27 ± 0.10	71.57%
P.C.	4.02 ± 0.13	2.88 ± 0.10	71.48%
P.R.E.	3.39 ± 0.04	2.42 ± 0.03	71.52%
qasper	1.52 ± 0.12	1.09 ± 0.09	71.65%
qmsum	3.50 ± 0.14	2.50 ± 0.10	71.57%
hotpotqa	3.66 ± 0.13	2.62 ± 0.09	71.52%
lcc	1.04 ± 0.10	0.75 ± 0.07	71.75%
R.P.	3.30 ± 0.20	2.36 ± 0.14	71.52%
samsum	2.71 ± 0.17	1.94 ± 0.13	71.60%
trec	1.70 ± 0.09	1.22 ± 0.07	71.66%
triviaqa	3.01 ± 0.19	2.16 ± 0.13	71.56%
RiddleBench	0.28 ± 0.00	0.21 ± 0.00	<u>73.58%</u>
SimpleMath	0.27 ± 0.00	0.19 ± 0.00	71.86%
AIME25	0.29 ± 0.00	0.21 ± 0.00	73.25%
AIME24	0.29 ± 0.00	0.21 ± 0.00	73.13%
LCB	0.41 ± 0.01	0.30 ± 0.00	72.63%
mbpp	0.27 ± 0.00	0.19 ± 0.00	72.46%
Avg.	2.10 ± 0.09	1.50 ± 0.07	71.94%

Table 4: Memory usage of DeepSeek-V2-Lite comparing MLA and compressed cache, with compression ratio shown. Standard error is noted in the lower right corner; boldface and underline mark the lowest and highest ratios, respectively.

Extension to MLA. MLA (Multi-head Latent Attention) is a mechanism designed to reduce the memory footprint of the KV cache. Our method can be combined with MLA to achieve further compression. We evaluate this combination on various datasets by compressing both the RoPE embeddings and the latent vectors stored in the cache. The results on DeepSeek-V2-Lite are shown in Table 4. The baseline memory refers to the original storage of RoPE and latent values without compression (denoted as “MLA” in the table). The column “Comp.” shows the memory usage after applying our compression method, and “Ratio” indicates the compressed size as a percentage of the baseline (lower is better). On average, our method reduces the memory footprint to 71.94% of the original MLA cache, with consistent performance across datasets. This demonstrates that our approach can complement existing KV cache compression techniques such as MLA.

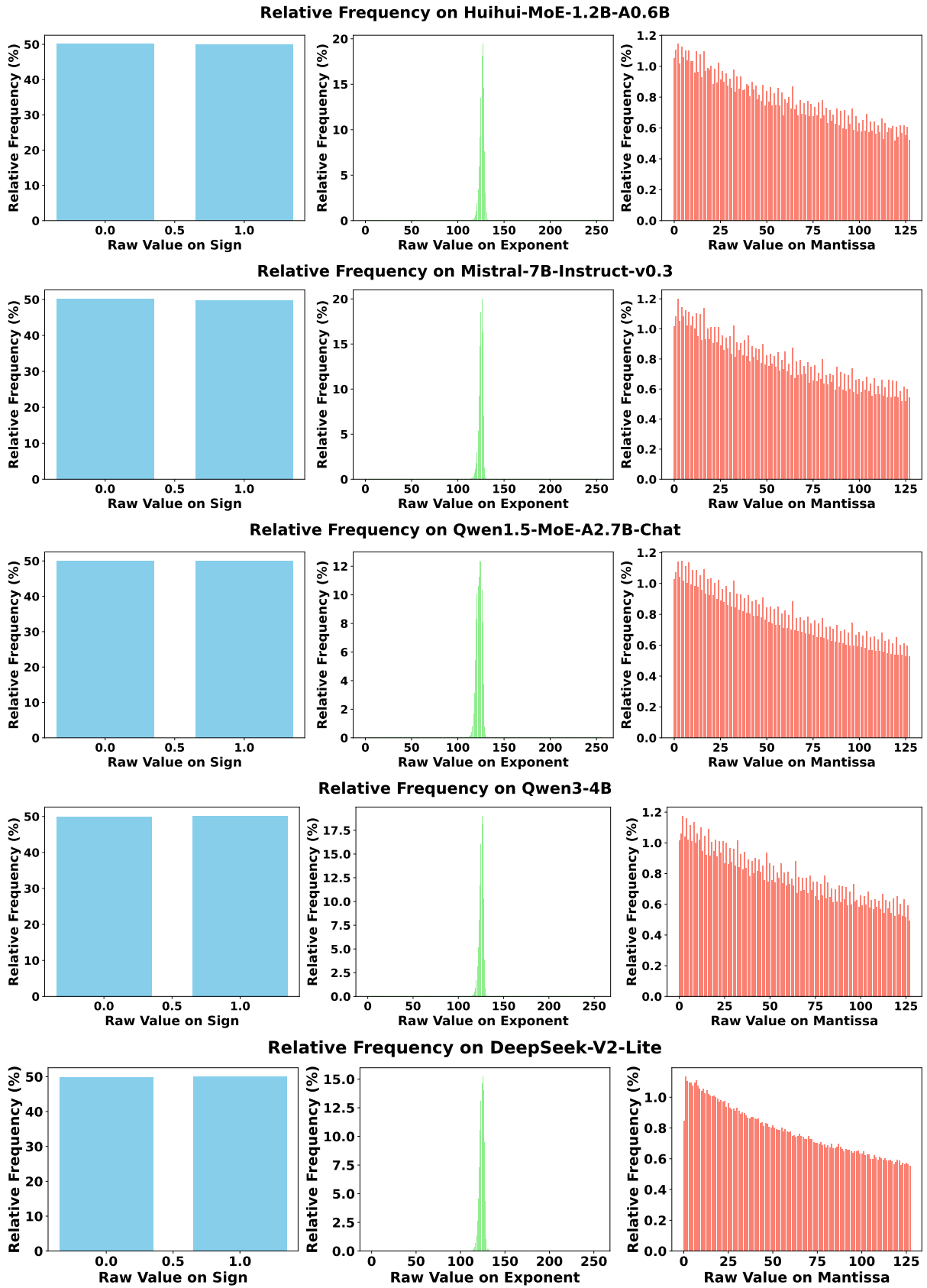


Figure 11: Relative frequency distribution of sign, exponent, and mantissa values in BFloat16 KV cache across various large language models.

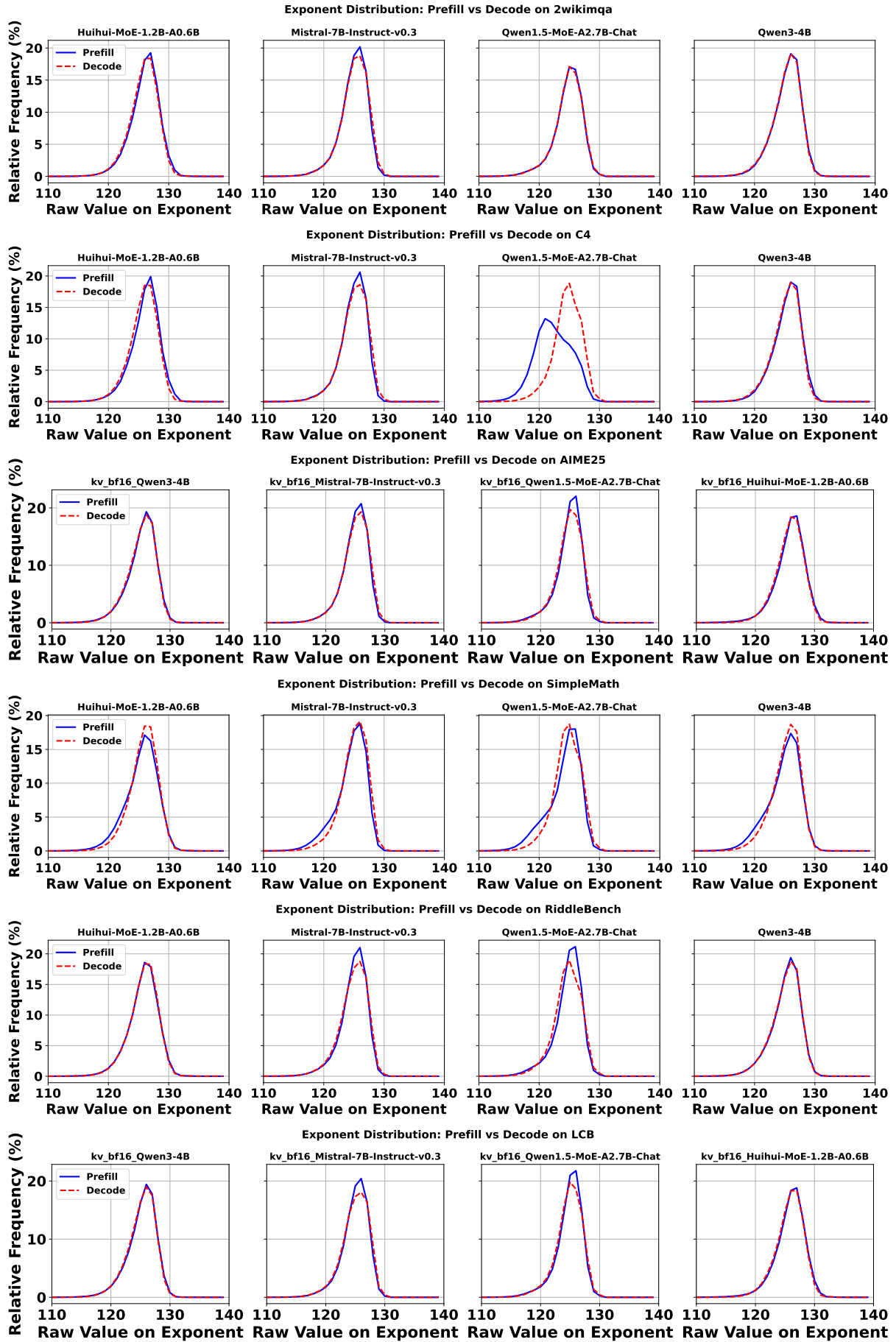


Figure 12: Comparison of exponent distribution across datasets for prefill and decode stages.

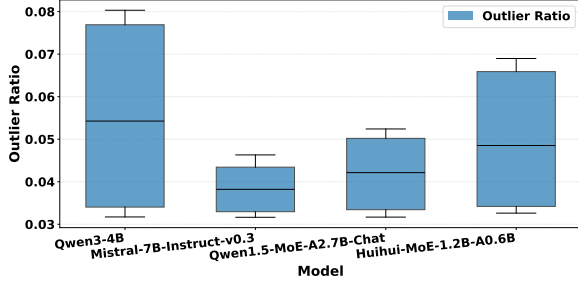


Figure 13: Distribution of outlier ratios in the KV cache across four different LLMs.

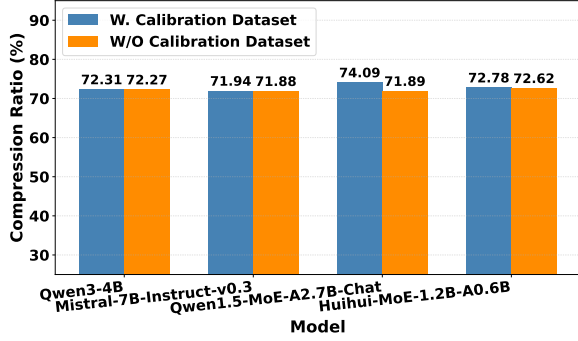


Figure 14: Impact of calibration dataset on compression ratio on multi_news dataset.

Model	BF16	FP16	E4M3	E5M2
Qwen3-4B	73.36	92.38	92.18	80.32
Mistral-7B	72.67	91.48	90.51	78.97
Qwen1.5-MoE	73.13	91.93	90.80	79.49
Huihui-MoE	73.71	92.82	93.42	81.20

Table 6: Compression ratio (%) on different data formats across four models.

Model	Setting	α	KS distance
Huihui-MoE-1.2B-A0.6B	Prefill	1.632	0.0239
	Decode	1.589	0.0128
Mistral-7B-Instruct-v0.3	Prefill	1.751	0.0137
	Decode	1.741	0.0091
Qwen1.5-MoE-A2.7B-Chat	Prefill	1.408	0.0521
	Decode	1.673	0.0104
Qwen3-4B	Prefill	1.697	0.0189
	Decode	1.660	0.0121

Table 7: α -stable fitting results on C4. Lower distance indicate better fit.

Extension to Other Data Formats. Our method can be extended to other floating-point formats. To provide a preliminary validation, we apply it to Float16 (FP16) and two FP8 variants (E4M3, E5M2) on C4 dataset. Since our hardware does not natively support FP8, we simulate the KV cache in FP16 and then convert it to the target FP8 formats

Dataset	Orig. (GB)	Comp. (GB)	Ratio (%)
Accounting	0.43 $\pm$ 0.05	0.3 $\pm$ 0.03	70.28
Agriculture	1.55 $\pm$ 0.03	1.09 $\pm$ 0.02	70.24
A.E.	0.83 $\pm$ 0.02	0.58 $\pm$ 0.02	70.27
Art	0.56 $\pm$ 0.01	0.4 $\pm$ 0.01	70.21
Art_Theory	0.39 $\pm$ 0.01	0.28 $\pm$ 0.01	70.26
B.M.S.	0.68 $\pm$ 0.48	0.48 $\pm$ 0.33	70.25
Biology	0.28 $\pm$ 0.03	0.2 $\pm$ 0.02	70.25
Chemistry	0.9 $\pm$ 0.61	0.63 $\pm$ 0.43	70.29
Clinical_Medicine	0.25 $\pm$ 0.03	0.17 $\pm$ 0.02	70.26
Computer_Science	0.74 $\pm$ 0.18	0.52 $\pm$ 0.13	70.25
Design	0.28 $\pm$ 0.04	0.19 $\pm$ 0.03	70.25
D.L.M.	0.22 $\pm$ 0.03	0.15 $\pm$ 0.02	70.28
Economics	0.72 $\pm$ 0.38	0.51 $\pm$ 0.27	70.27
Electronics	1.38 $\pm$ 0.39	0.97 $\pm$ 0.27	70.27
Energy_and_Power	2.07 $\pm$ 0.20	1.45 $\pm$ 0.14	70.27
Finance	1.0 $\pm$ 0.56	0.7 $\pm$ 0.39	70.28
Geography	0.28 $\pm$ 0.08	0.2 $\pm$ 0.06	70.26
History	0.37 $\pm$ 0.03	0.26 $\pm$ 0.02	70.26
Literature	0.33 $\pm$ 0.07	0.23 $\pm$ 0.05	70.24
Manage	0.57 $\pm$ 0.24	0.4 $\pm$ 0.17	70.27
Marketing	0.73 $\pm$ 0.17	0.52 $\pm$ 0.12	70.28
Materials	0.85 $\pm$ 0.06	0.59 $\pm$ 0.04	70.26
Math	0.51 $\pm$ 0.07	0.36 $\pm$ 0.05	70.26
M.E.	0.74 $\pm$ 0.32	0.52 $\pm$ 0.23	70.25
Music	1.38 $\pm$ 0.63	0.97 $\pm$ 0.44	70.27
Pharmacy	0.36 $\pm$ 0.03	0.25 $\pm$ 0.02	70.27
Physics	0.28 $\pm$ 0.02	0.19 $\pm$ 0.02	70.25
Psychology	0.41 $\pm$ 0.04	0.29 $\pm$ 0.03	70.26
Public_Health	0.23 $\pm$ 0.02	0.16 $\pm$ 0.02	<u>70.31</u>
Sociology	0.31 $\pm$ 0.03	0.22 $\pm$ 0.02	70.26
Avg.	0.65 $\pm$ 0.08	0.46 $\pm$ 0.06	70.26

Table 5: Memory usage of Qwen3-VL-8B-Instruct comparing original and compressed KV cache, with compression ratio (Comp./Orig.) shown. Standard error is noted in the lower right corner; boldface and underline mark the lowest and highest ratios, respectively.

to compute the compression ratio. Table 6 shows the results. While BFloat16 (BF16) achieves the lowest compression ratio (best compression) because of its higher exponent bit ratio, the method also works on other formats. We emphasize that the main focus of this paper is lossless KV cache compression under BF16, which is the most widely used format in recent LLMs. The results above are provided solely to demonstrate the generalization of the core idea and to encourage future exploration.

Distribution Analysis of KV-cache. In practice, the distribution of KV cache can be approximated by an α -stable distribution. We fit α -stable distributions to BF16 KV-cache histograms from the C4 dataset, covering both prefill and decode phases. Table 7 reports the fitting details. The estimated α values range from 1.408 to 1.751 across different models and settings, with small fitting errors (KS

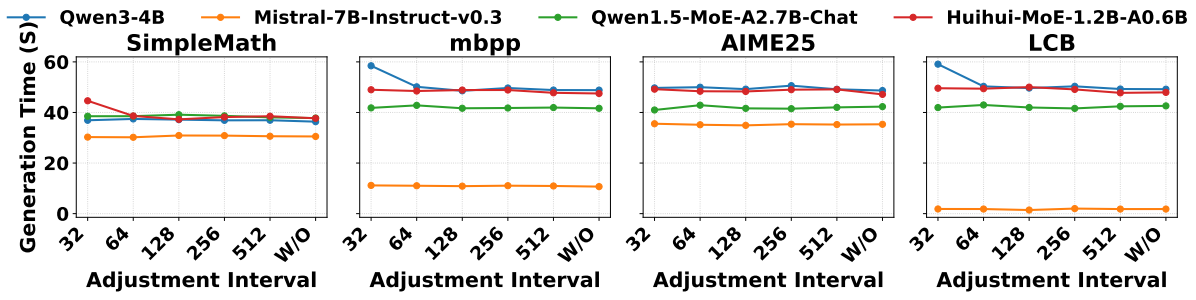


Figure 15: Generation time impact of different adjustment intervals. ‘W/O’ means without the adjuster.

distance). These results suggest that the α -stable distribution serves as a reasonable empirical model for KV-cache values.