LOCE: LOCALITY-AWARE LOSSLESS COMPRESSION FOR EFFICIENT LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Lossless compression offers a promising direction for alleviating the memory and bandwidth bottlenecks in LLMs due to inherent floating-point exponent redundancy. We observe that this redundancy primarily arises from extreme **value locality** and **temporal locality**, yet existing methods fail to unify these two forms of locality, leaving significant compression potential untapped. To bridge this gap, we propose **Loce**, a locality-aware lossless compression framework that leverages an algorithm-system co-design. At the algorithmic level, we present a fixed-length Rice-based lossless coding scheme that transforms both value and temporal locality into compact residual representations for lightweight encoding and decoding. At the system level, we introduce a GPU-efficient design with optimized memory layout, decoding kernel, and inference pipeline to accelerate compressed LLM inference. Extensive experiments on various LLMs and floating-point precisions show that, beyond achieving competitive compression ratios, Loce delivers an average 2.3× kernel-level acceleration and 1.5× end-to-end speedup over previous lossless compression methods. Our code is available on Anonymous GitHub.

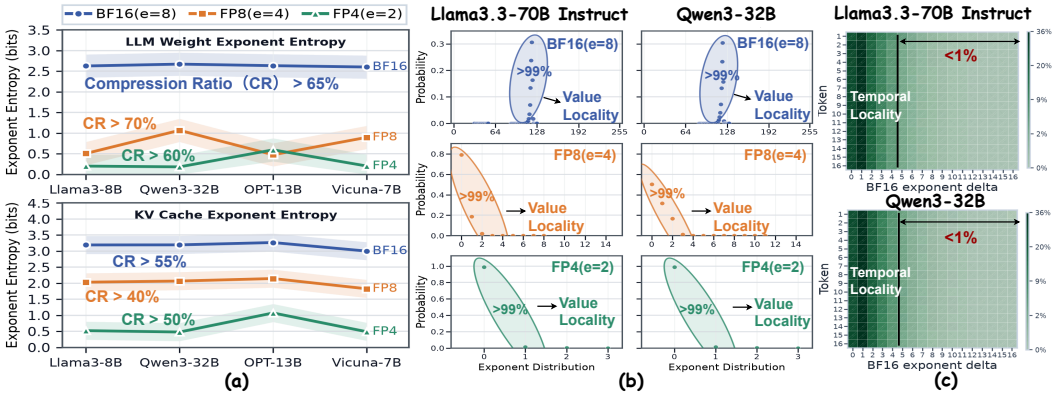

Figure 1: (a) Exponent entropy and Compression Ratio (CR) of LLM weights and KV cache tensors under BF16, FP8, and NVFP4 formats. (b) Value locality of weight exponents. (c) Temporal locality of KV cache exponents across tokens. With a 64K sequence length and 16-token KV pages in vLLM (Kwon et al., 2023), over 99% of BF16 KV cache exponent deltas, defined as the absolute exponent differences relative to the first token, are within 4 on WikiText-2 (Merity et al., 2016).

1 INTRODUCTION

As model scaling continues to drive remarkable advances in the generalization and reasoning capabilities of large language models (LLMs), the escalating memory footprint has become a critical barrier for LLM deployment, particularly in resource-constrained edge environments. Consequently, model compression has emerged as a key technique for alleviating the substantial resource demands of LLMs. However, most existing model compression approaches (e.g., quantization (Liang et al., 2026) and pruning (Yang et al., 2026c)) are lossy. For instance, aggressive 4-bit quantization (e.g., BNB-nf4 (Dettmers et al., 2023)) can incur up to 59% performance degradation on long-context tasks, while even 8-bit GPTQ-int8 (Frantar et al., 2022) quantization causes an 11.1% drop on NOCHA (Mekala et al., 2025). Such accuracy degradation can pose significant risks in high-stakes applications, including medical decision support and safety-critical agents (Xiang et al., 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Lossless compression (Zhang et al., 2026) is suggested as a promising solution to reduce model size without compromising model accuracy. The potential for lossless compression stems from the redundancy in the exponent fields of floating-point data (Heilper & Singer, 2025). As shown in Figure 1(a), our analysis across multiple LLMs reveals substantial redundancy in the exponent fields of model weights and KV cache, where the exponent entropy is 40%–70% lower than the allocated bit widths. To understand the source of redundancy, we carefully analyze the exponent distributions in various LLMs and identify a key observation: ***exponent distributions exhibit significant locality***. In particular, we characterize the exponent locality in model weight and KV cache as two forms:

❶ **Value Locality:** LLM weights often exhibit approximately zero-mean normal-like distributions (Si et al., 2025), causing exponents to concentrate within a narrow contiguous range, as illustrated in Figure 1(b). The theoretical analysis is provided in Appendix A.1.
❷ **Temporal Locality:** KV cache exponents change slowly across consecutive tokens. Temporal locality is intuitively related to the contextual continuity among neighboring tokens, which tend to produce similar KV magnitudes and exponent patterns. Within each KV page (16 tokens), the KV cache exponents of subsequent tokens remain close to the first token's exponents, as shown in Figure 1(c). Extended results are presented in Appendix A.2.

This key observation motivates the central question of our work: ***Can lossless compression jointly exploit value locality and temporal locality for efficient LLM inference?***

However, answering this central question remains challenging. Several lossless compression approaches employ entropy coding schemes, such as Huffman coding (Agrawal et al., 2026) and Asymmetric Numeral Systems (ANS) (Ibrahim et al., 2026), to achieve higher compression ratios. Nevertheless, such entropy coding schemes rely on variable-length bitstreams and complex indexing, which induce irregular memory accesses and hinder GPU parallelism, increasing latency by up to 3× (Yang et al., 2026a; Tan et al., 2026). Meanwhile, recent compression frameworks (Fan et al., 2026) leverage value locality in model weights to improve inference efficiency, but overlook temporal locality in KV cache tensors, limiting the potential benefits from exponent redundancy.

In this paper, we propose `Loce`, a lossless compression framework that exploits exponent locality to compress both LLM weights and KV cache tensors. ***Our key insight is that value and temporal locality can be transformed into concentrated residual distributions, enabling efficient lossless compression with lightweight Rice-based coding.*** Furthermore, `Loce` introduces system-level optimizations across memory layout, decoding kernel, and inference pipeline, effectively translating compression benefits into improved end-to-end LLM inference performance.

Contributions. ❶ **Unified Insight.** We identify exponent locality as the key commonality between LLM weights and KV cache tensors, providing a new perspective for efficient lossless compression. ❷ **Practical Solution.** `Loce` co-designs a lightweight fixed-length Rice-based coding scheme with GPU-efficient system optimizations, enabling lossless compression with accelerated LLM inference performance. ❸ **Empirical Study.** We comprehensively evaluate `Loce` against state-of-the-art lossless compression methods, including nvCOMP (Shanbhag et al., 2022), ZipNN (Hershcovitch et al., 2025), NeuZip (Hao et al., 2024), DFloat11 (Zhang et al., 2026), ECF8 (Yang et al., 2026b), and ZipServ (Fan et al., 2026). Extensive experiments demonstrate that `Loce` reduces up to 60 GB of memory footprint in long-context inference, achieves over 2.3× kernel-level acceleration, and delivers 1.5× average end-to-end speedup across diverse LLMs and numerical precisions.

2 Related Work

Lossy Model Compression. Existing model compression techniques broadly encompass pruning, knowledge distillation, and quantization (Zhu et al., 2024). Pruning (Chen et al., 2026) reduces model complexity by removing redundant parameters or structures. Knowledge distillation (Agarwal et al., 2024) transfers knowledge from a teacher model to a smaller student model. Quantization (Xiao et al., 2023) reduces storage and computational overhead by lowering the numerical precision of weights and activations. Nevertheless, these compression techniques are inherently lossy, modifying model parameters, structures, or numerical representations. In contrast, `Loce` reduces model size while preserving model accuracy, and can be combined with existing low-bit floating-point quantization methods to achieve additional compression gains, as discussed in Appendix D.4.

Table 1: Comparison with existing lossless model compression methods.

Method	Encoding Scheme	Code Structure	Supported Precision	Value Locality	Temporal Locality	Compressed Components
ZipNN (Hershcovitch et al., 2025)	Huffman Coding	Variable	FP32/BF16	✗	✗	Weight
NeuZip (Hao et al., 2024)	ANS	Variable	BF16	✗	✗	Weight
DFloat11 (Zhang et al., 2026)	Huffman Coding	Variable	BF16	✗	✗	Weight
ECF8 (Yang et al., 2026b)	Huffman Coding	Variable	FP8	✗	✗	Weight
ZipServ (Fan et al., 2026)	Triple Bitmap	Fixed	BF16	✓	✗	Weight
Loce (Ours)	**Rice Coding**	**Fixed**	**BF16/FP8/FP4**	✓	✓	**Weight + KV Cache**

* FP8 adopts the E4M3 format, while FP4 uses the NVFP4 format. Rice-based coding is designed based on standard Rice coding.

Lossless Model Compression. As summarized in Table 1, recent studies have proposed several lossless compression methods that leverage the low entropy of floating-point exponents. ZipNN (Hershcovitch et al., 2025), NeuZip (Hao et al., 2024), DFloat11 (Zhang et al., 2026), and ECF8 (Yang et al., 2026b) employ variable-length entropy coding schemes, such as Huffman coding and ANS, for weight compression, resulting in irregular memory accesses and significant runtime overhead. ZipServ (Fan et al., 2026) leverages value locality and adopts a fixed-length bitmap for model weights, but does not consider KV cache compression. SplitZip (Guo & Joshi, 2026) explores value locality-based lossless compression for KV cache transfer between prefill and decode workers in PD-disaggregated serving. However, the optimization in SplitZip primarily targets KV migration overhead and provides limited benefits for conventional single-node continuous batching inference without KV transfer. In contrast, Loce introduces a unified formulation of weight value locality and KV temporal locality, coupled with fixed-length Rice coding and system co-design.

3 METHOD

We present `Loce`, an algorithm-system co-designed framework with fixed-length Rice-based coding and system optimizations for efficient LLM inference. The main components are introduced as follows: Section 3.1 discusses the motivation of Rice-based coding, Section 3.2 presents the fixed-length Rice-based lossless compression scheme, and Section 3.3 describes the system design.

3.1 MOTIVATION: RICE CODING FOR EXPONENT COMPRESSION

Rice Coding Preliminaries. Rice coding is a lightweight lossless compression scheme designed for non-negative integers with highly skewed distributions (Sugiura et al., 2018). Given an integer x and a Rice-based parameter k, Rice coding decomposes x into a quotient q and a remainder r as

$$\underbrace{q = \left\lfloor \frac{x}{2^k} \right\rfloor, \qquad r = x \bmod 2^k}_{\text{ENCODE}} \quad \Longleftrightarrow \quad \underbrace{x = q \cdot 2^k + r}_{\text{DECODE}}. \tag{1}$$

The quotient q is encoded using unary coding, while the remainder r is represented using exactly k bits. For example, given $x = 13$ and $k = 2$, we obtain $q = 3$ and $r = 1$. Thus, q is encoded as 1110 and r as 01, yielding the Rice codeword 111001. Decoding reconstructs the original value as $3 \times 2^2 + 1 = 13$. This quotient-remainder decomposition requires only bit shifts and masks when 2^k is used as the divisor, enabling lightweight encoding and decoding for concentrated small values.

Why Rice Coding for Exponent Compression? Rice coding is particularly well suited to exponent compression because the observed exponent locality naturally converts into small integer residuals.

As shown in Table 2, for value locality, over 96% of BF16 weight exponents are concentrated within a contiguous range of seven values. Therefore, subtracting a shared base exponent produces small-magnitude residuals for most weights. For temporal locality, over 99% of KV cache exponents within a KV page differ from the first token by no more than seven, so using the first token as a base similarly produces small temporal residuals. With a fixed remainder width k, smaller residuals produce shorter Rice codes, allowing both value and temporal locality to be effectively exploited.

Table 2: BF16 exponent locality in model weights and KV cache across various LLMs.

Model	Precision	VL (≤ 7)	TL (≤ 7)
OPT-13B	BF16	96.3%	99.98%
Mistral-24B	BF16	97.4%	99.97%
Qwen3-32B	BF16	96.7%	99.98%
Llama3.3-70B Instruct	BF16	96.2%	99.99%

VL(≤ 7): coverage within a contiguous range of ≤ 7 values.
TL(≤ 7): coverage with $|\Delta E| \leq 7$ relative to the first token within each KV page.

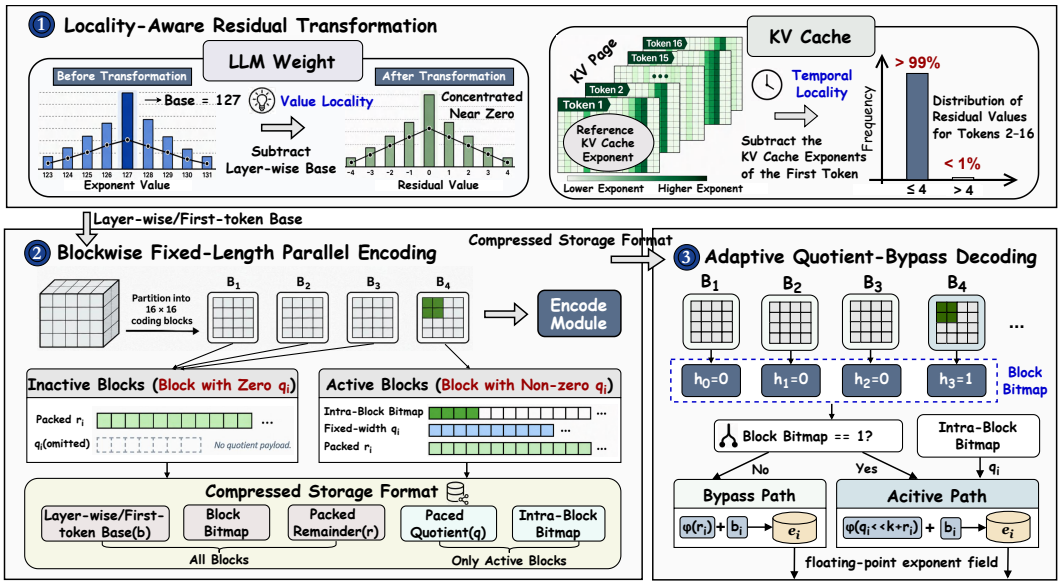

Figure 2: Overview of the fixed-length Rice-based lossless coding scheme.

Compared with Other Entropy Coding Schemes. While Huffman coding and ANS can achieve code lengths closer to the entropy bound, the highly skewed exponent distributions in LLMs make the resulting compression advantage over Rice coding relatively small. For example, for a representative exponent-residual distribution in which 95% of the residuals are concentrated within seven consecutive values, the measured average code length is 2.89 bits per symbol for Rice coding, closely approaching the 2.85 bits achieved by Huffman coding. In contrast, Rice coding provides a lightweight and regular computation pattern by avoiding complex codebook construction and table-based decoding, requiring only simple shift and arithmetic operations that are well suited for parallel GPU execution. Moreover, we replace the variable-length unary quotient in standard Rice coding with a fixed-length representation to enable regular memory access, as described in Section 3.2.

3.2 Fixed-Length Rice-Based Lossless Coding Scheme

Overview. We propose a fixed-length Rice-based lossless coding scheme, which transforms both value and temporal locality into a Rice-friendly residual distribution and employs fixed-width Rice coding to achieve compact encoding and efficient decoding. The overall coding scheme of `Loce` is illustrated in Figure 2 and the detailed algorithm pseudocode is provided in Appendix B.2.

Locality-Aware Residual Transformation. `Loce` first exploits the strong exponent locality to construct a compact base and transform the original exponents into small residuals. **For model weight tensors**, we employ a layer-wise statistical base. Specifically, for each layer l, the most frequent exponent is selected as the base, and the corresponding residual x_w is computed as:

$$c_l(e) = \sum_{i \in \mathcal{W}_l} \mathbf{1}(e_i = e), \qquad b_l = \arg\max_e c_l(e), \qquad x_w = e_i - b_l. \tag{2}$$

where $\mathcal{W}_l$ denotes the set of weight elements in layer l. Since weight exponents cluster around dominant values, the resulting residuals are largely concentrated near zero. **For KV cache**, `Loce` exploits the temporal locality across consecutive tokens by grouping every $P = 16$ tokens into a KV page (Kwon et al., 2023). Within each KV page, `Loce` retains the exponent vector of the first token as the page-level base and computes the exponent residuals of the remaining tokens as:

$$b^{kv}_{p,d} = e_{p,1,d}, \qquad x_{p,t,d} = e_{p,t,d} - b^{kv}_{p,d}, \quad t = 2, \ldots, P. \tag{3}$$

where p, t, and d denote the page index, token position, and feature dimension, respectively. The transformation converts exponent locality into small-magnitude integer residuals that are well suited for Rice coding. Moreover, since the residuals can be either positive or negative, `Loce` maps the signed residuals to non-negative integers through a reversible transformation:

$$z_i = \phi(x_i) = \begin{cases} 2x_i, & x_i \geq 0, \\ -2x_i - 1, & x_i < 0, \end{cases} \qquad \phi : \mathbb{Z} \to \mathbb{N}_0. \tag{4}$$

Blockwise Fixed-Length Parallel Encoding. After residual transformation, Loce packs the transformed residuals into a contiguous residual stream and partitions them into non-overlapping coding blocks, with each block containing 256 residuals and logically organized as 16×16. The Rice-based parameter k determines the remainder width. Within each coding block, every transformed residual z_i is independently decomposed into a quotient q_i and a remainder r_i:

$$q_i = z_i \gg k, \qquad r_i = z_i \,\&\, (2^k - 1). \tag{5}$$

For the quotient q_i, Loce replaces variable-length unary coding with a fixed-length representation, **while avoiding quotient storage for zero-quotient blocks**. Since the transformed residuals are highly concentrated near zero, most values satisfy $z_i < 2^k$ and produce zero quotients. For BF16 KV cache tensors, nearly 90% of coding blocks contain only zero quotients, with detailed statistics reported in Appendix B.1 (Tables 5–7). Based on this observation, Loce introduces a bitmap-guided quotient scheme and each coding block $\mathcal{B}_j$ is associated with a block indicator h_j:

Table 3: Configuration of Loce encoding under different precisions.

Datatype	Rice-based Parameter k	Remainder Width b_r	Quotient Width b_q
BF16	3	3 bits	6 bits
FP8(E4M3)	2	2 bits	3 bits
NVFP4	1	1 bit	2 bit

$$h_j = \mathbf{1}[\exists\, i \in \mathcal{B}_j : q_i \neq 0], \quad g_{j,i} = \mathbf{1}[q_i \neq 0],\ i \in \mathcal{B}_j, h_j = 1, \quad o_j^q = \left(\sum_{m<j} \sum_{i \in \mathcal{B}_m} g_{m,i} \right) b_q. \tag{6}$$

where $h_j = 0$ indicates that all quotients within the block are zero, while $h_j = 1$ activates the intra-block quotient bitmap $g_{j,i}$ to locate non-zero quotients. The corresponding quotient payload offset o_j^q is determined by the number of preceding non-zero quotients, with each quotient represented using the fixed width b_q. Based on the block indicator h_j, Loce handles each block in two forms:

- $h_j = 0$**:** Only the packed remainder stream is stored, while the quotient payload is omitted.
- $h_j = 1$**:** The block stores the quotient bitmap and non-zero quotients using b_q bits in Table 3.

Adaptive Quotient-Bypass Decoding. Loce decoding follows the blockwise encoding layout. For each coding block $\mathcal{B}_j$, the decoder first checks the block indicator h_j. When $h_j = 0$, all quotients are zero and z_i is reconstructed from the remainder. For active blocks, the decoder uses the intra-block quotient bitmap $g_{j,i}$ to identify stored quotients. The decoding process is formulated as follows:

$$z_i = \begin{cases} r_i, & h_j g_{j,i} = 0, \\ (q_i \ll k) + r_i, & h_j g_{j,i} = 1, \end{cases} \quad x_i = \phi^{-1}(z_i) = \begin{cases} z_i/2, & z_i \bmod 2 = 0. \\ -(z_i+1)/2, & z_i \bmod 2 = 1. \end{cases} \tag{7}$$

The original exponent is then recovered as $e_i = b_i + x_i$, where b_i denotes the layer-wise base for model weights and the page-level base for KV cache tensors. Combining the reconstructed exponent with the unchanged sign and mantissa fields restores the original floating-point value.

Compression Ratio Analysis. Loce stores a fixed-width remainder for each exponent residual and allocates quotient storage only to non-zero positions. As a result, its storage cost is governed by the zero-quotient block ratio and the density of non-zero quotients within active blocks. For BF16 weights in LLaMA3.3-70B, the measured statistics correspond to a compression ratio of around $1.4\times$, while KV cache achieves a comparable ratio with only a small amortized overhead from page-level bases. A detailed analysis and the compression bounds are provided in Appendix B.3.

3.3 System Design and Implementation

Loce adopts a **load-compressed, compute-decompressed** design (Fan et al., 2026). For model weights, Loce performs offline encoding and decodes weights on demand during inference. For KV cache, continuously encoding newly generated KV entries during autoregressive decoding incurs substantial runtime overhead. Therefore, Loce adopts a trade-off strategy: **compress the prefill-generated KV cache once, while keeping newly generated KV entries uncompressed.** This strategy provides substantial memory savings (Appendix D.1), while requiring only one parallel encoding process instead of repeated encoding during autoregressive decoding. Meanwhile, to minimize the introduced decoding overhead and maximize the performance benefits from reduced memory traffic, Loce introduces three effective system-level optimization techniques:

❶ **Aligned Compressed Memory Layout.** Loce introduces a byte-aligned memory layout based on the 16×16 coding block granularity, reducing cross-boundary memory accesses and alignment overheads to enable efficient GPU loading of compressed tensors. Specifically, Loce organizes each coding block into four regions: the remainder stream, block bitmap, intra-block bitmap, and quotient payload. The remainder stream uses a fixed k-bit representation, resulting in predictable byte-aligned storage (e.g., $16 \times 16 \times 3 = 768$ bits (96 bytes) for BF16). The block bitmap is byte-packed across coding blocks, while the intra-block bitmap contains $16 \times 16 = 256$ bits per coding block and is stored as a 32-byte aligned bitmap. The quotient payload uses fixed-width storage with block-level padding for byte alignment, incurring negligible padding overhead, while bitmap-based rank operations directly determine quotient positions without scanning preceding codewords.

❷ **Warp-Cooperative Adaptive Decoding Kernel.** Loce designs a warp-cooperative decoding kernel by mapping each 16×16 coding block to one GPU warp. During decoding, the block bitmap is frequently accessed for quotient payload selection, and Loce caches packed block bitmaps in registers to accelerate path selection. For each coding block containing 256 exponent residuals, 32 threads in one warp collaboratively reconstruct the block, with each thread responsible for eight residual elements through static index mapping. The remainder stream is evenly distributed across warp lanes, allowing each thread to independently extract its assigned fixed-width remainders without bitstream parsing. For zero-quotient blocks, the kernel bypasses quotient memory accesses and directly reconstructs residuals from the remainder stream. For active blocks, warp-level ballot and population-count operations are used to process the intra-block bitmap, compute quotient ranks, and directly locate fixed-width quotient entries. The decoded residuals are reconstructed in registers before exponent restoration, avoiding intermediate global memory writes.

❸ **Memory-Compute Overlapped Inference Pipeline.** Sequential compressed block loading and decoding limit overall throughput by leaving memory transfer and computation on the critical path. To hide memory-access latency, Loce employs a double-buffered inference pipeline that overlaps compressed-data prefetching with warp-level decoding, as shown in Figure 3. The fixed-size remainder regions and predictable block metadata enable Loce to determine the addresses of upcoming coding blocks before decoding completes. Specifically, two shared-memory buffers are used in a ping-pong manner: while one buffer supplies the block for warp-level decoding, the other asynchronously prefetches the remainder stream and metadata of the next block from global memory. Once the block indicator is available, quotient data are fetched only for active blocks. After decoding the current block, the two buffers exchange roles, allowing memory transfer for the next block to overlap with computation on the current block.

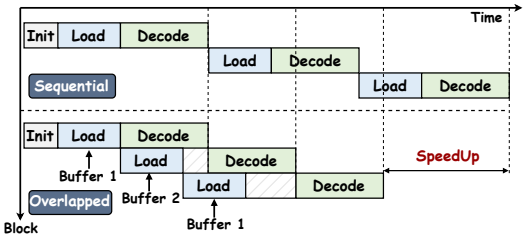

Figure 3: Overview of the double-buffered memory-compute inference pipeline that overlaps compressed-data loading with decoding.

4 Experiments

4.1 Experimental Setup

Datasets and Workloads. ❶ Datasets: We evaluate Loce on two representative long-context benchmarks, RULER-QA (Hsieh et al., 2024) and LongBench (Bai et al., 2024). RULER-QA supports controllable sequence length through configurable distractor passages, while LongBench provides 21 real-world long-context tasks spanning QA, summarization, few-shot learning, synthetic reasoning, and code completion. ❷ Workloads: We choose four widely used LLMs, including OPT (Zhang et al., 2022), Mistral (Jiang et al., 2023), Qwen3 (Yang et al., 2025), and LLaMA-3.3 (Grattafiori et al., 2024) for evaluation. We further apply Loce to compress the weights of diffusion models, including Wan2.2 (Wan et al., 2025) and Qwen-Image (Wu et al., 2025), as shown in Appendix D.3. Details of datasets and workloads are provided in Appendix C.1 and Appendix C.2.

Baselines. We compare Loce with six lossless compression methods, including NeuZip (Hao et al., 2024), nvCOMP (Shanbhag et al., 2022), ZipNN (Hershcovitch et al., 2025), DFloat11 (Zhang et al., 2026), ZipServ (Fan et al., 2026), and ECF8 (Yang et al., 2026b). These methods target lossless

Table 4: Comparison of compression ratio and memory footprint on representative LLMs with batch size 4 and sequence length 64K on RULER-QA.

Model	Method	Precision	Compression Ratio		Memory Footprint(GB)		Avg. Bit Width		Memory Reduction (GB)
			Weight	KV Cache	Weight	KV Cache	Weight	KV Cache	
Mistral-24B	Uncompressed (BF16)	BF16	1.00	1.00	48.70	42.95	16.00	16.00	0.00
	nvCOMP	BF16	1.04	1.00	46.83	42.95	15.38	16.00	1.87
	ZipNN	BF16	1.48	1.00	32.91	42.95	10.81	16.00	15.79
	NeuZip	BF16	1.48	1.00	32.91	42.95	10.81	16.00	15.79
	DFloat11	BF16	1.44	1.00	33.82	42.95	11.11	16.00	14.88
	ZipServ	BF16	1.27	1.00	38.35	42.95	12.60	16.00	10.35
	Loce (BF16)	**BF16**	**1.41**	**1.38**	**34.54**	**31.12**	**11.35**	**11.59**	**25.99**
	Uncompressed (FP8)	FP8	1.00	1.00	24.72	21.33	8.00	8.00	0.00
	ECF8	FP8	1.24	1.00	19.94	21.33	6.45	8.00	4.78
	Loce (FP8)	**FP8**	**1.27**	**1.27**	**19.46**	**16.80**	**6.30**	**6.30**	**9.79**
	Uncompressed (NVFP4)	NVFP4	1.00	1.00	13.61	11.86	4.50	4.50	0.00
	Loce (NVFP4)	**NVFP4**	**1.24**	**1.22**	**10.98**	**9.72**	**3.63**	**3.69**	**4.77**
Qwen3-32B	Uncompressed (BF16)	BF16	1.00	1.00	65.80	69.10	16.00	16.00	0.00
	nvCOMP	BF16	1.03	1.00	63.88	69.10	15.53	16.00	1.92
	ZipNN	BF16	1.49	1.00	44.16	69.10	10.74	16.00	21.64
	NeuZip	BF16	1.47	1.00	44.76	69.10	10.88	16.00	21.04
	DFloat11	BF16	1.40	1.00	47.00	69.10	11.43	16.00	18.80
	ZipServ	BF16	1.29	1.00	51.01	69.10	12.40	16.00	14.79
	Loce (BF16)	**BF16**	**1.38**	**1.33**	**47.68**	**51.95**	**11.59**	**12.03**	**35.27**
	Uncompressed (FP8)	FP8	1.00	1.00	32.18	33.67	8.00	8.00	0.00
	ECF8	FP8	1.20	1.00	26.82	33.67	6.67	8.00	5.36
	Loce (FP8)	**FP8**	**1.28**	**1.19**	**25.14**	**28.29**	**6.25**	**6.72**	**12.42**
	Uncompressed (NVFP4)	NVFP4	1.00	1.00	19.04	20.13	4.50	4.50	0.00
	Loce (NVFP4)	**NVFP4**	**1.25**	**1.21**	**15.23**	**16.64**	**3.60**	**3.72**	**7.30**
Llama-3.3-70B Instruct	Uncompressed (BF16)	BF16	1.00	1.00	141.10	88.34	16.00	16.00	0.00
	nvCOMP	BF16	1.03	1.00	137.03	88.34	15.53	16.00	4.07
	ZipNN	BF16	1.49	1.00	94.71	88.34	10.75	16.00	46.39
	NeuZip	BF16	1.48	1.00	95.39	88.34	10.82	16.00	45.71
	DFloat11	BF16	1.42	1.00	99.40	88.34	11.27	16.00	41.70
	ZipServ	BF16	1.27	1.00	111.15	88.34	12.60	16.00	29.95
	Loce (BF16)	**BF16**	**1.39**	**1.36**	**101.43**	**64.91**	**11.50**	**11.75**	**63.10**
	Uncompressed (FP8)	FP8	1.00	1.00	63.76	42.95	8.00	8.00	0.00
	ECF8	FP8	1.17	1.00	54.69	42.95	6.86	8.00	9.07
	Loce (FP8)	**FP8**	**1.27**	**1.22**	**50.20**	**35.20**	**6.30**	**6.56**	**21.31**
	Uncompressed (NVFP4)	NVFP4	1.00	1.00	40.35	27.18	4.50	4.50	0.00
	Loce (NVFP4)	**NVFP4**	**1.26**	**1.23**	**32.03**	**22.10**	**3.57**	**3.66**	**13.40**

*nvCOMP adopts LZ4 compression, while ZipServ uses its kernel-based implementation. Memory Reduction represents the total memory saving over the uncompressed baseline, including both weights and KV cache. All LLMs use grouped-query attention (GQA). NVFP4 adopts block-wise scaling, and the scaling overhead is included in the reported memory footprint and average bit width.

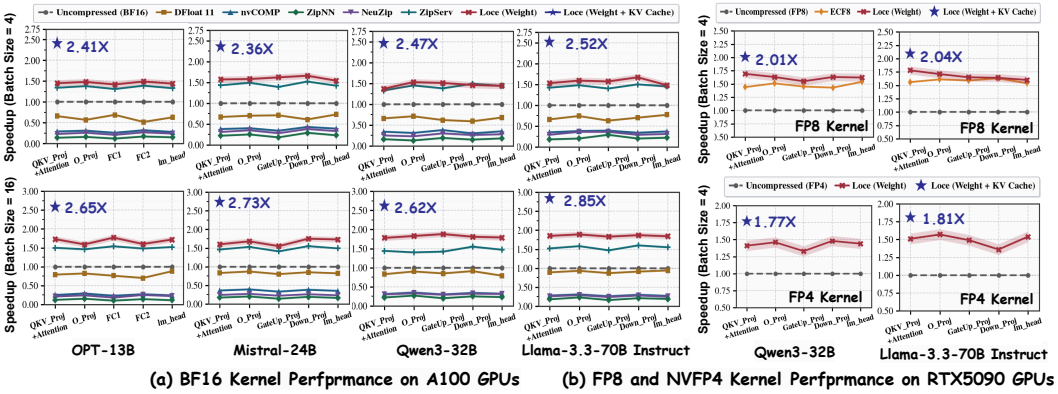

Figure 4: Kernel performance comparison with a sequence length of 64K.

weight compression, whereas Loce supports lossless compression of both model weights and KV cache tensors generated during the prefill stage. Please refer to the Appendix C.3 for more details.

Experiment Setting. We implement Loce with PyTorch (Paszke et al., 2019), CUDA (Guide, 2020), and vLLM (Kwon et al., 2023) for LLM inference evaluation on two GPU platforms: an 8× NVIDIA A100 (80GB) GPU server and a 4× NVIDIA RTX 5090 GPU server. Detailed hardware and software configurations are provided in Appendix C.4. FP8 uses the E4M3 format and FP4 adopts NVFP4. BF16 experiments are conducted on A100 GPUs, while FP8 and FP4 experiments are conducted on RTX 5090 GPUs for native low-precision support. For kernel-level performance evaluation, all measurements are repeated 20 times, and the average latency is reported. For end-to-end inference evaluation, we measure the average latency after a 100-step warm-up period. All baseline methods are evaluated under the same experimental settings to ensure fair comparisons.

4.2 Compression and Kernel Performance Evaluation

Compression Ratio and Memory Footprint. Table 4 shows the compression ratio and memory footprint of lossless compression methods on various LLMs. Loce consistently reduces memory

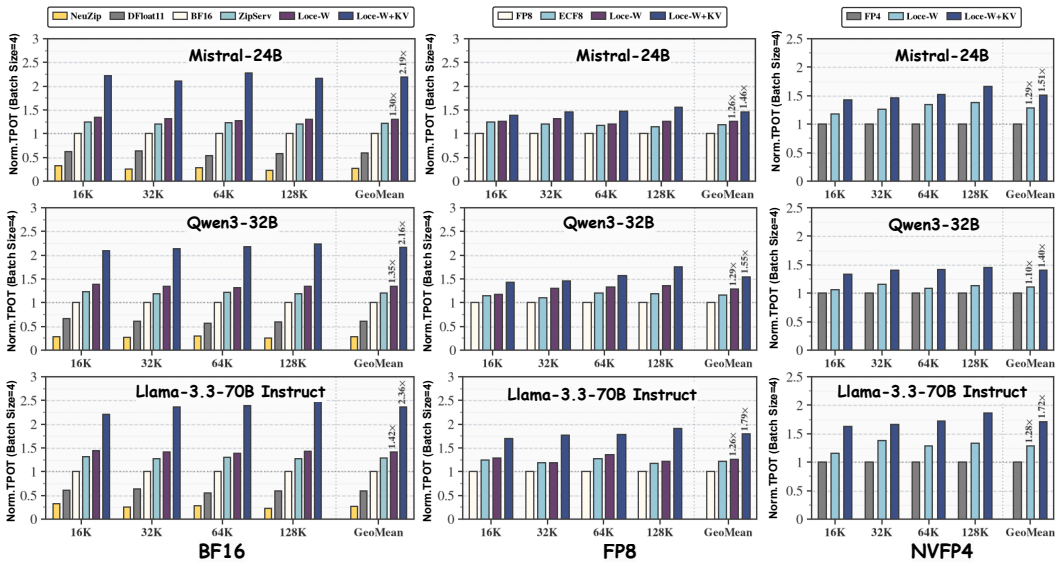

Figure 5: Time-Per-Output-Token (TPOT) comparison on the LongBench benchmark.

footprint across different LLMs and numerical precisions by compressing model weights and prefill-generated KV cache tensors, achieving up to 63.10 GB memory reduction and compression ratios close to the theoretical estimates in the Appendix B.3. Although Loce achieves a slightly lower weight compression ratio than DFloat11 with Huffman coding (less than 0.03×), the gap mainly comes from the fixed-width coding and alignment padding adopted to support efficient execution.

Kernel Performance. We evaluate major kernels with integrated decoding under different lossless compression methods on RULER-QA, including QKV projection, attention, output projection, and feed-forward projections, as shown in Figure 4. The execution time of QKV projection and attention operations is reported together. Loce achieves kernel performance improvements over existing lossless compression methods by employing a compact Rice-based coding scheme and GPU-efficient decoding optimizations. The compression of weights benefits all evaluated kernels by reducing memory traffic, while KV cache compression mainly improves the performance of the attention operation. Complex variable-length coding schemes introduce high decoding overhead and may even degrade performance below the uncompressed baseline. Moreover, as the batch size increases, Loce achieves further performance gains due to better amortization of decoding overhead.

4.3 End-to-end Inference Performance

TPOT. Figure 5 shows the TPOT (Zhong et al., 2024) of different lossless compression methods under various numerical precisions on LLMs. Since the decode stage of LLM inference is primarily memory-bound (Shah et al., 2024), Loce improves inference efficiency by reducing weight and KV cache memory traffic while introducing only low-overhead decompression operations. As a result, Loce achieves up to 2.36× TPOT improvement compared with existing lossless compression methods. Moreover, as the sequence length increases, the growing memory footprint of weights and KV cache further amplifies the benefits of compression, leading to larger performance gains.

Breakdown Analysis. Figure 6 illustrates the latency breakdown of the LLM inference pipeline on the LongBench benchmark, including loading, prefill, decode, and the one-time encoding step for the prefill-generated KV cache. The overall latency breakdown also reflects the time-to-first-token (TTFT), which consists of the loading and prefill stages. We select an uncompressed BF16 model as a baseline for comparison. Loce slightly reduces the loading overhead by weight compression. Although Loce introduces an additional one-time KV cache en-

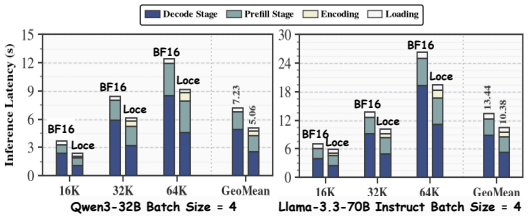

Figure 6: Latency breakdown of the LLM inference.

coding step after prefill, the incurred overhead remains small and has limited impact on overall latency. The prefill stage incurs similar overhead before and after compression. Notably, the decode stage becomes the dominant bottleneck for long sequence length inference. As the sequence length increases, the growing KV cache footprint further amplifies the decode overhead, which accounts for over 75% of the total latency under the 64K sequence length setting. By reducing weight and KV cache transfer time during decoding, Loce improves end-to-end inference efficiency.

4.4 Ablation Study

Since Loce preserves model accuracy (as detailed in Appendix D.2), the ablation study focuses on analyzing the contributions of different design choices to LLM inference performance.

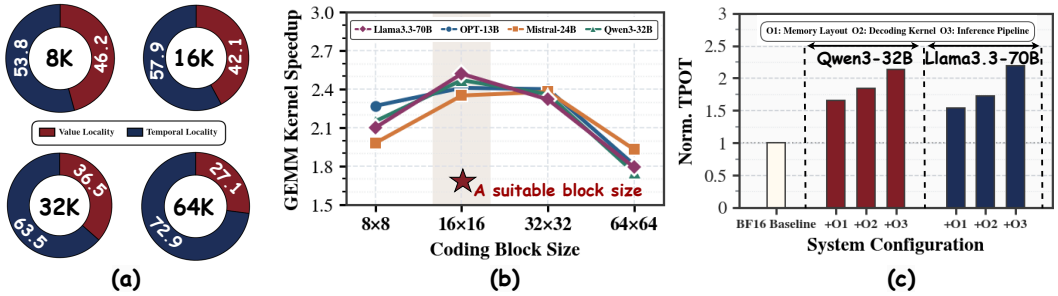

Figure 7: Ablation studies on key design choices of Loce.

Ablation on Value and Temporal Locality. Figure 7(a) investigates the impact of value locality and temporal locality on the achieved speedup across different sequence lengths. At 8K context length, both locality patterns provide comparable benefits. However, as the sequence length grows, temporal locality contributes increasingly to the overall acceleration and becomes the dominant factor, demonstrating the importance of exploiting temporal locality in long-context inference.

Ablation on Coding Block Size. We conduct an ablation study on the coding block size used in the encoding and decoding process described in Section 3.2. As shown in Figure 7(b), different block sizes exhibit a trade-off between compression efficiency and decoding performance. Smaller block sizes introduce higher metadata overhead and limit the exploitation of zero-quotient blocks, while larger block sizes reduce decoding parallelism due to the increased probability of activating quotient storage. Based on this trade-off, we select a 16×16 coding block as the default configuration.

Ablation on System Optimization. We evaluate the effectiveness of the system optimizations introduced in Section 3.3, including the aligned compressed memory layout (O1), warp-cooperative adaptive decoding kernel (O2), and memory-compute overlapped inference pipeline (O3). As shown in Figure 7(c), O1 provides the largest TPOT improvement by directly optimizing compressed memory access and reducing memory traffic. This benefit becomes more pronounced under long-context workloads, where decoding is increasingly dominated by memory access overhead. O2 further improves performance by optimizing decoding execution on GPU compute cores through warp-level cooperation, while O3 enhances on-chip storage utilization by overlapping memory access and computation. The combination of all optimizations achieves the best performance, highlighting the importance of hardware-aware optimization for efficient lossless LLM compression.

5 Conclusion

In this paper, we propose Loce, an algorithm-system co-designed lossless compression framework for efficient LLM inference. Our core contribution lies in identifying exponent locality as a unified source of redundancy across LLM weights and KV cache tensors, and exploiting both value and temporal locality for lossless compression. We further introduce a fixed-length Rice-based coding scheme, together with GPU-efficient optimizations including aligned memory layout, optimized decoding kernels, and an overlapped inference pipeline. Extensive experiments across various LLMs and numerical precisions demonstrate that Loce achieves substantial memory savings and accelerates LLM inference without sacrificing accuracy, highlighting its potential for LLM deployment.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Aditya Agrawal, Albert Magyar, Hiteshwar Eswaraiah, Patrick Sheridan, Pradeep Janedula, Ravi Krishnan Venkatesan, Krishna Nair, and Ravi Iyer. Single-stage huffman encoder for ml compression. *arXiv preprint arXiv:2601.10673*, 2026.

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. Longbench: A bilingual, multitask benchmark for long context understanding, 2024.

Inesh Chakrabarti, David Limpus, Aditi Ghai Rana, Bowen Bao, Spandan Tiwari, Thiago Crepaldi, and Ashish Sirasao. Ultraquant: 4-bit kv caching for context-heavy agents. *arXiv preprint arXiv:2606.20474*, 2026.

Weiyu Chen, Hansi Yang, Yunhao Gou, Han Shi, Enliang Hu, Zhenguo Li, and James Kwok. Multi-objective one-shot pruning for large language models. *Advances in Neural Information Processing Systems*, 38:91960–91992, 2026.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 1286–1305, 2021.

Ruibo Fan, Xiangrui Yu, Xinglin Pan, Zeyu Li, Weile Luo, Qiang Wang, Wei Wang, and Xiaowen Chu. Zipserv: Fast and memory-efficient llm inference with hardware-aware lossless compression. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 2264–2280, 2026.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew

Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha,

Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024.

Design Guide. Cuda c++ programming guide. *NVIDIA, July*, pp. 6–11, 2020.

Yipin Guo and Siddharth Joshi. Splitzip: Ultra fast lossless kv compression for disaggregated llm serving. *arXiv preprint arXiv:2605.01708*, 2026.

Yongchang Hao, Yanshuai Cao, and Lili Mou. Neuzip: Memory-efficient training and inference with dynamic compression of neural networks. *arXiv preprint arXiv:2410.20650*, 2024.

Anat Heilper and Doron Singer. Lossless compression of neural network components: Weights, checkpoints, and k/v caches in low-precision formats. *arXiv preprint arXiv:2508.19263*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Moshik Hershcovitch, Andrew Wood, Leshem Choshen, Guy Girmonsky, Roy Leibovitz, Ilias Ennmouri, Michal Malka, Peter Chin, Swaminathan Sundararaman, and Danny Harnik. Zipnn: Lossless compression for ai models, 2025.

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

Kareem Ibrahim, Mohammadjavad Maheronnaghsh, and Andreas Moshovos. Shannonic: Efficient entropy-optimal compression for ml workloads. *Proceedings of Machine Learning and Systems*, 8:1515–1529, 2026.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Yesheng Liang, Haisheng Chen, Song Han, and Zhijian Liu. Paroquant: Pairwise rotation quantization for efficient reasoning llm inference. In *International Conference on Learning Representations*, volume 2026, pp. 77000–77017, 2026.

Anmol Mekala, Anirudh Atmakuru, Yixiao Song, Marzena Karpinska, and Mohit Iyyer. Does quantization affect models' performance on long-context tasks? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9433–9481, 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

Anil Shanbhag, Bobbi W Yogatama, Xiangyao Yu, and Samuel Madden. Tile-based lightweight integer compression in gpu. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1390–1403, 2022.

Chongjie Si, Jingjing Jiang, and Wei Shen. Unveiling the mystery of weight in large foundation models: Gaussian distribution never fades. *arXiv preprint arXiv:2501.10661*, 2025.

Ryosuke Sugiura, Yutaka Kamamoto, Noboru Harada, and Takehiro Moriya. Optimal golomb-rice code extension for lossless coding of low-entropy exponentially distributed sources. *IEEE Transactions on Information Theory*, 64(4):3153–3161, 2018.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024.

Hongshi Tan, Yao Chen, Gustavo Alonso, Weng-Fai Wong, and Bingsheng He. Approaching shannon bound with lossless llm weight compression. *arXiv preprint arXiv:2606.15789*, 2026.

Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, Jiayu Wang, Jingfeng Zhang, Jingren Zhou, Jinkai Wang, Jixuan Chen, Kai Zhu, Kang Zhao, Keyu Yan, Lianghua Huang, Mengyang Feng, Ningyi Zhang, Pandeng Li, Pingyu Wu, Ruihang Chu, Ruili Feng, Shiwei Zhang, Siyang Sun, Tao Fang, Tianxing Wang, Tianyi Gui, Tingyu Weng, Tong Shen, Wei Lin, Wei Wang, Wei Wang, Wenmeng Zhou, Wente Wang, Wenting Shen, Wenyuan Yu, Xianzhong Shi, Xiaoming Huang, Xin Xu, Yan Kou, Yangyu Lv, Yifei Li, Yijing Liu, Yiming Wang, Yingya Zhang, Yitong Huang, Yong Li, You Wu, Yu Liu, Yulin Pan, Yun Zheng, Yuntao Hong, Yupeng Shi, Yutong Feng, Zeyinzi Jiang, Zhen Han, Zhi-Fan Wu, and Ziyu Liu. Wan: Open and advanced large-scale video generative models, 2025.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning, 2025.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang,

Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025.

Jinwu Yang, Jiaan Wu, Zedong Liu, Xinyang Ma, Hairui Zhao, Yida Gu, Yuanhong Huang, Xingchen Liu, Wenjing Huang, Zheng Wei, Jing Xing, Yili Ma, Qingyi Zhang, Baoyi An, Zhongzhe Hu, Shaoteng Liu, Xia Zhu, Jiaxun Lu, Guangming Tan, and Dingwen Tao. Enec: A lossless ai model compression method enabling fast inference on ascend npus, 2026a.

Zeyu Yang, Tianyi Zhang, Jianwen Xie, Chuan Li, Zhaozhuo Xu, and Anshumali Shrivastava. To compress or not? pushing the frontier of lossless genai model weights compression with exponent concentration. In *International Conference on Learning Representations*, volume 2026, pp. 53467–53487, 2026b.

Zhiguo Yang, Changjian Deng, Qinke Chen, Zijing Zhou, and Jian Cheng. Lsa: Layer-wise sparsity allocation for large language model pruning based on minimal linear reconstruction error. In *The Fourteenth International Conference on Learning Representations*, 2026c.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

Tianyi Zhang, Mohsen Hariri, Shaochen Henry Zhong, Vipin Chaudhary, Yang Sui, Xia Hu, and Anshumali Shrivastava. 70% size, 100% accuracy: Lossless llm compression for efficient gpu inference via dynamic-length float (dfloat11). *Advances in Neural Information Processing Systems*, 38:98966–98994, 2026.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12: 1556–1577, 2024.

Appendix

Table of Contents

A Theoretical Analysis and Extended Results of Exponent Locality 16

A.1 Theoretical Analysis of Value Locality in LLM Weight Exponents 16

A.2 Extended Results on Temporal Locality 17

B Additional Details of the Method 20

B.1 Exponent Locality and Zero-Quotient Block Statistic 20

B.2 Algorithm Pseudocode 20

B.3 Detailed Storage and Compression Ratio Analysis 21

C Experimental and Implementation Details 25

C.1 Dataset Details and Reproducibility 25

C.2 Generative AI Models Used for Evaluation 25

C.3 Lossless Compression Baselines 25

C.4 Software and Hardware 26

D Supplementary Experimental Results 26

D.1 Analysis of Prefill and Decode KV Cache Footprint 26

D.2 Accuracy Evaluation 27

D.3 Evaluation on Diffusion Transformers 28

D.4 Orthogonal to Low-Bit Floating-Point Quantization Techniques 28

E Limitation Discussion and Future Work 28

F Use of LLM 29

A Theoretical Analysis and Extended Results of Exponent Locality

A.1 Theoretical Analysis of Value Locality in LLM Weight Exponents

We theoretically analyze why LLM weight exponents exhibit strong value locality. Following the commonly adopted Gaussian approximation of LLM weights (Dettmers et al., 2023; Si et al., 2025; Fan et al., 2026), we assume that

$$W \sim \mathcal{N}(0, \sigma^2), \tag{8}$$

where $\sigma > 0$ denotes the layer-wise weight scale. For any nonzero weight, define the unbiased binary exponent as

$$X = \lfloor \log_2 |W| \rfloor, \qquad X = x \Longleftrightarrow 2^x \le |W| < 2^{x+1}. \tag{9}$$

A window of K consecutive exponents $\{\ell, \ell+1, \ldots, \ell+K-1\}$ corresponds to the magnitude interval $[2^\ell, 2^{\ell+K})$. Let $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and PDF of $\mathcal{N}(0,1)$, respectively. The probability mass covered by this exponent window is

$$\begin{aligned} P_K(\ell;\sigma) &= \Pr\left(\ell \le X \le \ell + K - 1\right) \\ &= \Pr\left(2^\ell \le |W| < 2^{\ell+K}\right) \\ &= 2\left[\Phi\left(\frac{2^{\ell+K}}{\sigma}\right) - \Phi\left(\frac{2^\ell}{\sigma}\right)\right]. \end{aligned} \tag{10}$$

Let $a = 2^\ell/\sigma$. Then

$$C_K(a) = 2\left[\Phi(2^K a) - \Phi(a)\right], \qquad a > 0. \tag{11}$$

The derivative is

$$\begin{aligned} C_K'(a) &= 2\left[2^K \phi(2^K a) - \phi(a)\right] \\ &= 2\phi(a)\left[2^K \exp\left(-\frac{(2^{2K}-1)a^2}{2}\right) - 1\right]. \end{aligned} \tag{12}$$

Since $\phi(a) > 0$ for $a > 0$, the stationary point satisfies

$$a_K^\star = \sqrt{\frac{2K \ln 2}{2^{2K}-1}}. \tag{13}$$

Moreover,

$$C_K'(a) > 0 \quad (a < a_K^\star), \qquad C_K'(a) < 0 \quad (a > a_K^\star), \tag{14}$$

indicating that $a_K^\star$ uniquely maximizes the probability mass captured by a K-exponent window.

Since ℓ is restricted to integers, we select

$$\ell^\star = \mathrm{round}\left(\log_2(\sigma a_K^\star)\right). \tag{15}$$

The rounding operation guarantees

$$\frac{a_K^\star}{\sqrt{2}} \le \frac{2^{\ell^\star}}{\sigma} \le \sqrt{2} a_K^\star, \tag{16}$$

and therefore

$$P_K(\ell^\star;\sigma) \ge \underbrace{\min\left\{C_K\left(\frac{a_K^\star}{\sqrt{2}}\right), C_K\left(\sqrt{2}a_K^\star\right)\right\}}_{\underline{C}_K}. \tag{17}$$

Importantly, $\underline{C}_K$ is independent of σ, indicating that exponent concentration is determined by the distribution shape rather than the absolute weight scale. Numerically,

$$\underline{C}_7 = 95.87\%, \qquad \underline{C}_9 = 98.36\%, \qquad \underline{C}_{11} = 99.32\%. \tag{18}$$

Thus, under the Gaussian approximation, most LLM weights are guaranteed to have exponents concentrated within a small contiguous range, which provides the theoretical foundation of value locality.

We further extend the analysis to finite-precision floating-point formats. Let $Q_{\mathcal{F}}$ denote round-to-nearest quantization to a binary floating-point format $\mathcal{F}$, and define

$$\widehat{W} = Q_{\mathcal{F}}(W), \qquad \widehat{X} = \left\lfloor \log_2 |\widehat{W}| \right\rfloor . \tag{19}$$

For finite normal values,

$$E_{\mathcal{F}} = \widehat{X} + b_{\mathcal{F}}, \tag{20}$$

where $b_{\mathcal{F}}$ denotes the exponent bias.

Assume that the exponent interval under consideration lies within the normal finite range of $\mathcal{F}$. Since powers of two are exactly representable and rounding-to-nearest can cross at most one adjacent exponent boundary,

$$2^{\ell} \le |W| < 2^{\ell+K} \Longrightarrow \widehat{X} \in \{\ell, \ell+1, \ldots, \ell+K\}. \tag{21}$$

Consequently,

$$E_{\mathcal{F}} \in \{\ell + b_{\mathcal{F}}, \ldots, \ell + K + b_{\mathcal{F}}\}. \tag{22}$$

Therefore, floating-point representation preserves exponent locality while introducing at most one additional adjacent exponent value.

For scaled low-precision formats, let

$$\widetilde{W} = \frac{W}{s}, \qquad s > 0. \tag{23}$$

Then

$$\widetilde{W} \sim \mathcal{N}\left(0, \frac{\sigma^2}{s^2}\right). \tag{24}$$

Because the concentration bound $\underline{C}_K$ is independent of the distribution scale, replacing σ with σ/s does not affect the exponent locality property. Hence, subject to the normal finite range of each format, the same exponent-concentration principle applies to BF16, FP8 (e.g., E4M3), and scaled FP4 (e.g., E2M1).

A.2 Extended Results on Temporal Locality

Temporal Locality Across Layers. We evaluate the temporal locality of KV cache exponents across layers in several LLMs, where all exponent deltas are defined as absolute differences. Figures 8, 9, and 10 show temporal locality across all layers of Llama3-8B, OPT-13B, and Qwen3-32B.

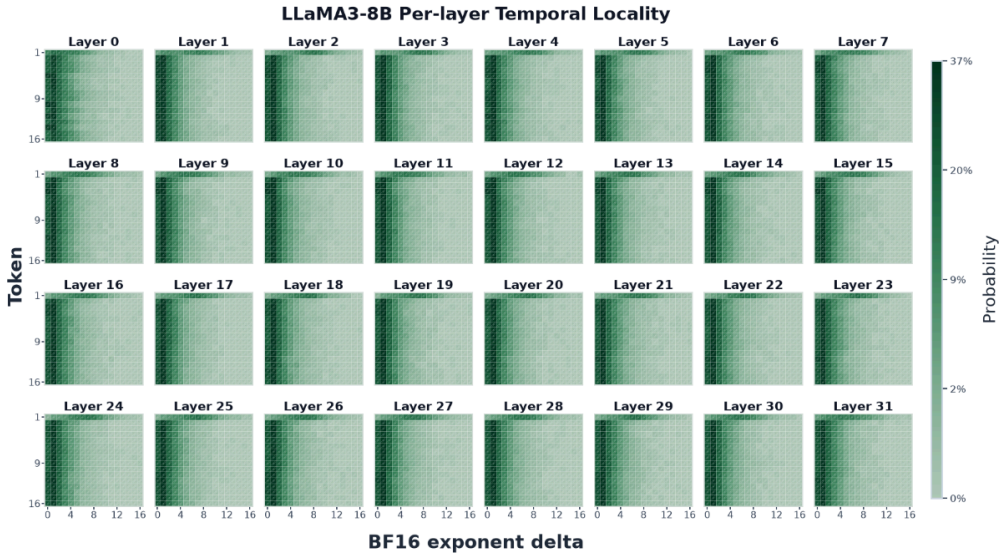

Figure 8: Temporal Locality of Per-Layer KV Cache in Llama3-8B.

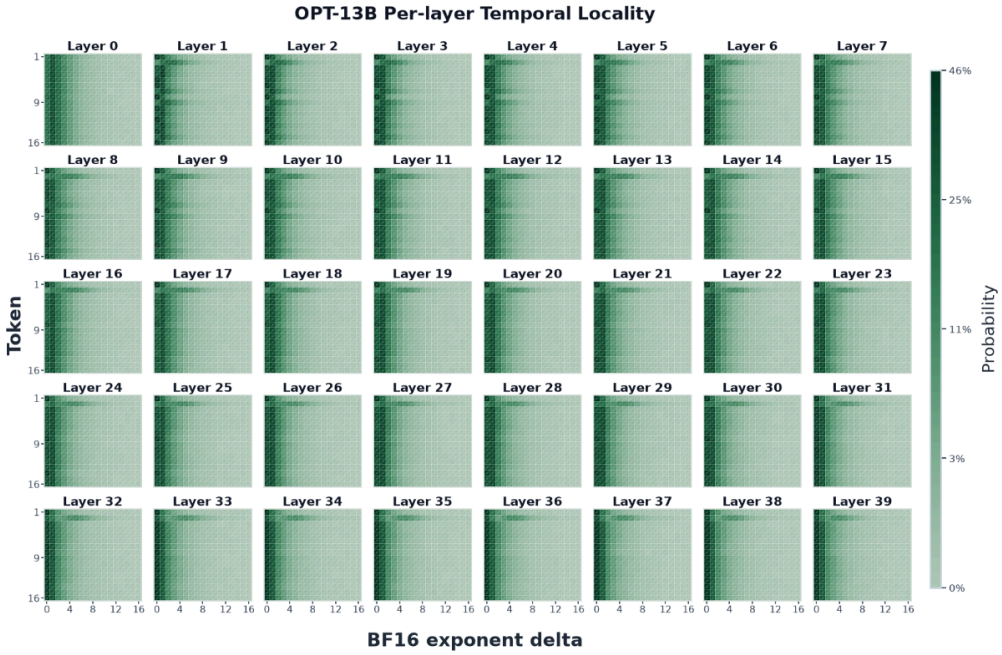

Figure 9: Temporal Locality of Per-Layer KV Cache in OPT-13B.

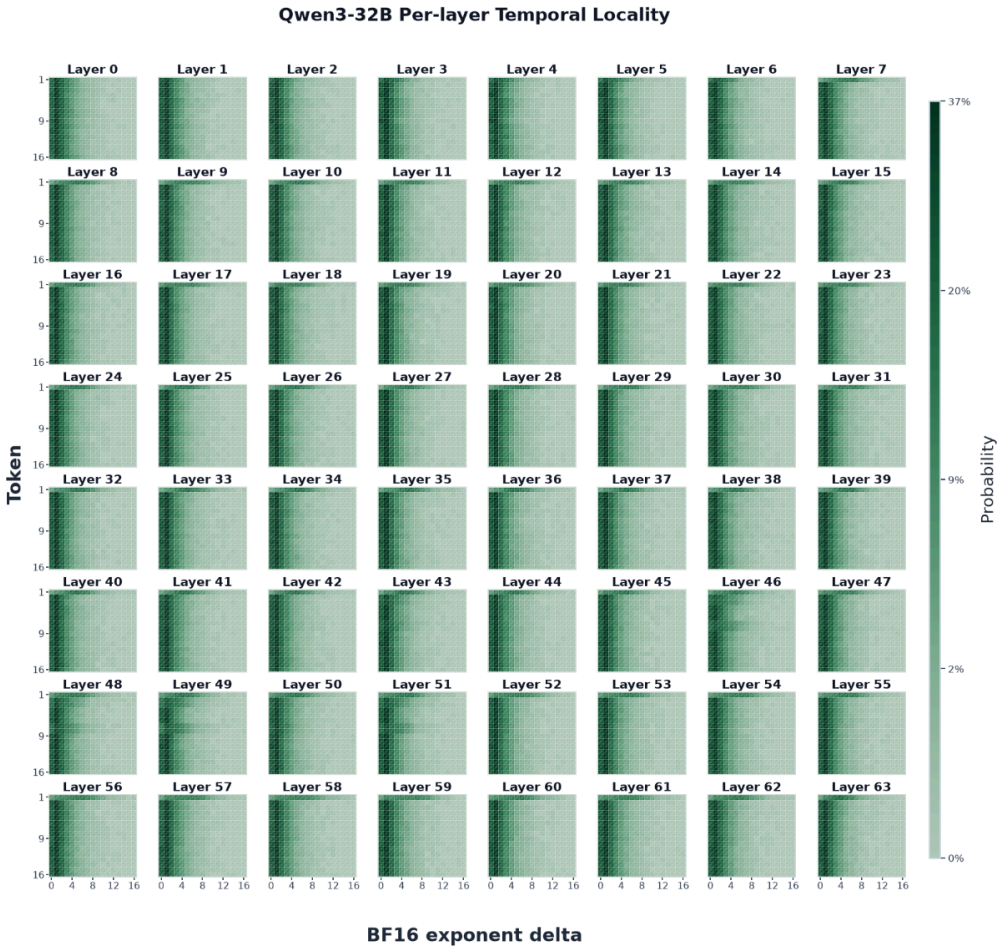

Figure 10: Temporal Locality of Per-Layer KV Cache in Qwen3-32B.

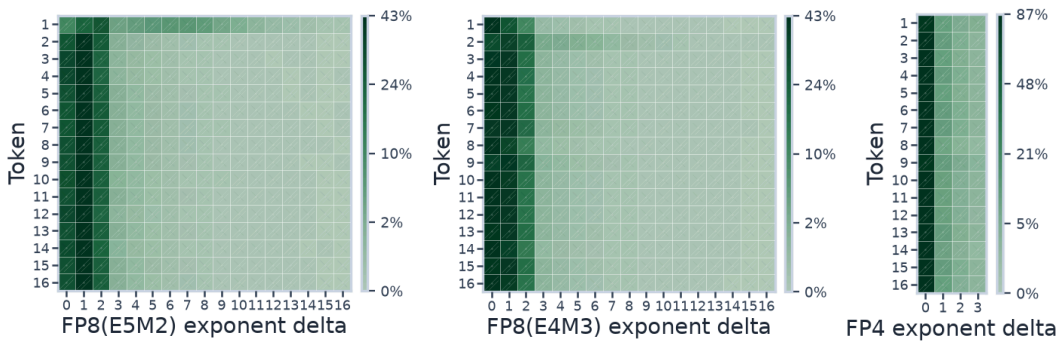

Figure 11: Temporal locality of KV cache exponents under various precisions on OPT-13B.

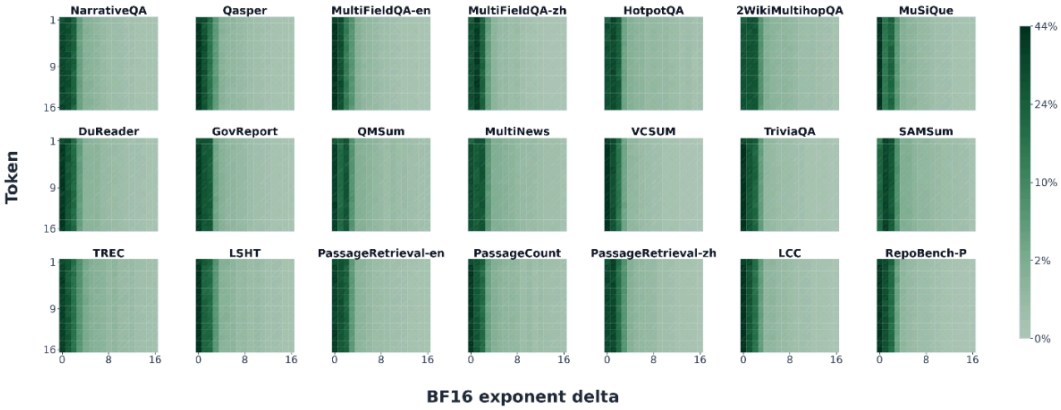

Figure 12: Temporal locality of KV Cache exponents across long-context workloads.

Table 5: Exponent locality and zero-quotient block statistics for BF16 tensors across various LLMs.

Model	Precision	Block Size	VL(≤ 7)	TL(≤ 7)	Weight Zero-Quotient Blocks	KV Zero-Quotient Blocks
OPT-13B	BF16	16×16	96.3%	99.98%	90.1%	95.7%
Mistral-24B	BF16	16×16	97.4%	99.97%	91.1%	96.5%
Qwen3-32B	BF16	16×16	96.7%	99.98%	90.5%	95.3%
LLaMA3.3-70B Instruct	BF16	16×16	96.2%	99.99%	90.3%	97.1%

Table 6: Exponent locality and zero-quotient block statistics for FP8 tensors across various LLMs.

Model	Precision	Block Size	VL(≤ 3)	TL(≤ 3)	Weight Zero-Quotient Blocks	KV Zero-Quotient Blocks
OPT-13B	FP8 (E4M3)	16×16	95.5%	99.31%	89.5%	92.7%
Mistral-24B	FP8 (E4M3)	16×16	96.1%	99.66%	89.9%	94.1%
Qwen3-32B	FP8 (E4M3)	16×16	96.5%	99.25%	90.3%	94.0%
LLaMA3.3-70B Instruct	FP8 (E4M3)	16×16	96.6%	99.82%	90.7%	96.3%

Table 7: Exponent locality and zero-quotient block statistics for FP4 tensors across various LLMs.

Model	Precision	Block Size	VL(≤ 1)	TL(≤ 1)	Weight Zero-Quotient Blocks	KV Zero-Quotient Blocks
OPT-13B	FP4	16×16	95.7%	98.65%	89.5%	92.1%
Mistral-24B	FP4	16×16	94.9%	98.54%	89.2%	94.0%
Qwen3-32B	FP4	16×16	95.9%	98.33%	90.1%	92.2%
LLaMA3.3-70B Instruct	FP4	16×16	96.8%	99.01%	91.0%	94.5%

VL: coverage within a contiguous range of seven exponent values.
TL: coverage of exponent differences relative to the first token within each KV page.
Weight Zero-Quotient Blocks: percentage of weight coding blocks in which all quotient values satisfy $q_i = 0$.
KV Zero-Quotient Blocks: percentage of KV cache coding blocks in which all quotient values satisfy $q_i = 0$.

Temporal Locality Across Precisions. We evaluate the temporal locality of KV cache exponents under different numerical formats. Figure 11 presents the temporal locality of OPT-13B across BF16, FP8, and FP4 formats, where all exponent deltas are defined as absolute differences. We observe that reducing the exponent bit width leads to smaller exponent variations within each KV page, making the temporal locality increasingly prominent in lower-precision formats.

Temporal Locality Across Datasets. As illustrated in Figure 12, we evaluate the temporal locality of KV cache exponents across diverse long-context workloads from LongBench (Bai et al., 2024), which includes 21 datasets spanning single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. Using LLaMA-3-8B under BF16 precision, we observe consistent temporal locality across all evaluated datasets, demonstrating that KV cache exponent locality persists across diverse task domains and input distributions.

B Additional Details of the Method

B.1 Exponent Locality and Zero-Quotient Block Statistic

To evaluate the exponent locality exploited by `Loce`, we report the detailed locality and zero-quotient block statistics across different LLMs and numerical precisions in Tables 5–7. The results show that exponent locality remains strong across BF16, FP8, and FP4 tensors, with most weight and KV cache exponents concentrated within a narrow range after residual transformation. Consequently, the transformed residuals produce a high proportion of zero-quotient blocks under the corresponding Rice configurations, validating the effectiveness of quotient omission and adaptive quotient-bypass decoding in `Loce`.

B.2 Algorithm Pseudocode

Algorithm 1 Fixed-Length Rice-Based Encoding

Input: Floating-point tensor $\mathbf{X}$; type $\tau \in \{\text{WEIGHT}, \text{KV}\}$; Rice-based parameter k; quotient width b_q; page size $P = 16$; block size $N_B = 256$ (16×16)
Output: Compressed representation $\mathcal{C}$

$\mathbf{X} \to (\mathbf{S}, \mathbf{E}, \mathbf{M})$ ▷ Extract floating-point fields
if $\tau = \text{WEIGHT}$ **then**
  for all layers l **do**
    $b_l \leftarrow \arg\max_e \sum_{i \in \mathcal{W}_l} \mathbf{1}(e_i = e)$; $x_i \leftarrow e_i - b_l,\ \forall i \in \mathcal{W}_l$ ▷ Layer-wise base
  end for
  $\mathcal{R} \leftarrow \{b_l\}_l$
else
  for all KV pages p **do**
    $b^{kv}_{p,d} \leftarrow e_{p,1,d}$; $x_{p,t,d} \leftarrow e_{p,t,d} - b^{kv}_{p,d},\ t = 2, \ldots, P$ ▷ Page-level exponent base
  end for
  $\mathcal{R} \leftarrow \{b^{kv}_{p,d}\}_{p,d}$
end if
$z_i \leftarrow \phi(x_i)$; partition $\{z_i\}$ into coding blocks $\{\mathcal{B}_j\}$ of size N_B ▷ Map signed residuals
for all $\mathcal{B}_j$ **in parallel do**
  $q_i \leftarrow z_i \gg k$, $r_i \leftarrow z_i \,\&\, (2^k - 1)$, $g_{j,i} \leftarrow \mathbf{1}[q_i \neq 0],\ \forall i \in \mathcal{B}_j$ ▷ Rice decomposition
  $h_j \leftarrow \mathbf{1}\left[\sum_{i \in \mathcal{B}_j} g_{j,i} > 0\right]$; $c_j \leftarrow \sum_{i \in \mathcal{B}_j} g_{j,i}$ ▷ Detect zero-quotient blocks
  $\text{Pack}_k\big(\{r_i\}_{i \in \mathcal{B}_j}\big) \to \mathcal{S}_r[jN_Bk : (j+1)N_Bk]$ ▷ Store fixed-width remainders
end for
$\mathbf{h} \leftarrow \{h_j\}_j$; $\mathcal{S}_g \leftarrow \text{Pack}(\{\mathbf{g}_j \mid h_j = 1\})$ ▷ Pack block and intra-block bitmaps
$\{o^q_j\}_j \leftarrow b_q\,\text{ExScan}(\{c_j\}_j)$ ▷ Compute quotient payload offsets
for all j with $h_j = 1$ **in parallel do**
  $\text{Pack}_{b_q}(\{q_i \mid g_{j,i} = 1,\ i \in \mathcal{B}_j\}) \to \mathcal{S}_q[o^q_j :]$ ▷ Pack non-zero quotients with fixed width
end for
$\mathcal{C} \leftarrow (\mathbf{S}, \mathbf{M}, \mathcal{R}, \mathbf{h}, \mathcal{S}_g, \mathcal{S}_r, \mathcal{S}_q)$ ▷ Assemble compressed representation
return $\mathcal{C}$

Algorithm 2 Fixed-Length Rice-Based Decoding

Input: Compressed representation $\mathcal{C}$; type $\tau \in \{\textsc{Weight}, \mathrm{KV}\}$; Rice-based parameter k; quotient width b_q
Output: Reconstructed floating-point tensor $\widehat{\mathbf{X}}$

$(\mathbf{S}, \mathbf{M}, \mathcal{R}, \mathbf{h}, \mathcal{S}_g, \mathcal{S}_r, \mathcal{S}_q) \leftarrow \mathcal{C}$ ▷ Parse compressed representation
for all $\mathcal{B}_j$ **in parallel do**
  $\{r_i\}_{i \in \mathcal{B}_j} \leftarrow \mathrm{Unpack}_k(\mathcal{S}_r)$ ▷ Load remainders
  if $h_j = 0$ **then**
    $z_i \leftarrow r_i,\ \forall i \in \mathcal{B}_j$ ▷ Bypass quotient decoding
  else
    Recover $\mathbf{g}_j$ and the stored non-zero quotients from $\mathcal{S}_g$ and $\mathcal{S}_q$ ▷ Load quotient payload
    Reconstruct $\{z_i\}_{i \in \mathcal{B}_j}$ from $\{r_i\}$, $\mathbf{g}_j$, and the recovered quotients ▷ Recover residuals
  end if
  $x_i \leftarrow \phi^{-1}(z_i),\ \forall i \in \mathcal{B}_j$ ▷ Recover signed residuals
end for
$\widehat{\mathbf{E}} \leftarrow \mathrm{RestoreExponent}(\{x_i\}, \mathcal{R}, \tau)$ ▷ Restore original exponents
$\widehat{\mathbf{X}} \leftarrow (\mathbf{S}, \widehat{\mathbf{E}}, \mathbf{M})$ ▷ Reassemble floating-point values
return $\widehat{\mathbf{X}}$

B.3 Detailed Storage and Compression Ratio Analysis

We provide a detailed analysis of the storage cost of `Loce` for model weights and KV cache tensors. The analysis accounts for the unchanged sign and mantissa fields, fixed-width remainder stream, block-level bitmap, intra-block quotient bitmap, non-zero quotient payload, and base exponents. We further derive compression-ratio bounds directly from the zero-quotient block statistics reported in Tables 5–7.

Storage Model. Consider a floating-point format with a total bit width B and an E-bit exponent field. Since `Loce` compresses only the exponent field, the remaining

$$U = B - E \tag{25}$$

sign-and-mantissa bits are preserved without modification. Each coding block contains $N_B = 256$ exponent residuals.

Let M denote the number of coding blocks. We define the zero-quotient block ratio as

$$\zeta = \frac{1}{M} \sum_{j=1}^{M} \mathbf{1}[h_j = 0], \tag{26}$$

such that $1 - \zeta$ is the fraction of active blocks containing at least one non-zero quotient.

We further define the global non-zero quotient ratio as

$$\rho_q = \frac{1}{M N_B} \sum_{j:h_j=1} \sum_{i \in \mathcal{B}_j} g_{j,i}, \tag{27}$$

which represents the fraction of all encoded residuals whose quotients are explicitly stored. Since each active block contains at least one and at most N_B non-zero quotients,

$$\frac{1-\zeta}{N_B} \le \rho_q \le 1 - \zeta. \tag{28}$$

Thus, the zero-quotient block ratio alone provides a strict bound on the quotient payload even when the exact value-level quotient sparsity is not explicitly reported.

Model Weight Storage. Consider a weight tensor containing N_w values and $M_w = N_w / N_B$ coding blocks. We assume that residuals are packed into a contiguous stream before block partitioning and that N_w is divisible by N_B. Padding of the final incomplete block, when needed, is negligible for large LLM weight tensors.

The unchanged sign and mantissa fields require

$$S_{\mathrm{sm}}^{w} = N_w U \tag{29}$$

bits, while the fixed-width remainder stream requires

$$S_{r}^{w} = N_w k \tag{30}$$

bits.

The block-level bitmap stores one bit h_j for each coding block, resulting in

$$S_{h}^{w} = M_w = \frac{N_w}{N_B} \tag{31}$$

bits.

For active blocks, `Loce` additionally stores an N_B-bit intra-block bitmap $\mathbf{g}_j$ to indicate the positions of non-zero quotients. Its total storage cost is therefore

$$\begin{aligned} S_{g}^{w} &= M_w(1-\zeta_w)N_B \\ &= N_w(1-\zeta_w). \end{aligned} \tag{32}$$

Only non-zero quotients are stored in the quotient payload. Let ρ_q^w denote the fraction of weight residuals with non-zero quotients. Since each quotient occupies b_q bits, the quotient payload requires

$$S_{q}^{w} = N_w \rho_q^w b_q. \tag{33}$$

Finally, let L denote the number of layer-wise base exponents. Their storage cost is

$$S_{\mathrm{ref}}^{w} = LE. \tag{34}$$

Combining all components gives

$$\begin{aligned} S_w = N_w U + N_w k + \frac{N_w}{N_B} \\ + N_w(1-\zeta_w) + N_w \rho_q^w b_q + LE. \end{aligned} \tag{35}$$

Dividing Eq. 35 by N_w, the average storage per weight is

$$\boxed{\bar{B}_w = U + k + \frac{1}{N_B} + (1-\zeta_w) + \rho_q^w b_q + \frac{LE}{N_w}.} \tag{36}$$

For large LLM weight tensors, the layer-base overhead LE/N_w is negligible. The corresponding compression ratio is

$$\boxed{\mathrm{CR}_w = \frac{B}{U + k + \frac{1}{N_B} + (1-\zeta_w) + \rho_q^w b_q + \frac{LE}{N_w}}.} \tag{37}$$

The memory-saving ratio is

$$\mathrm{SR}_w = 1 - \frac{\bar{B}_w}{B} = 1 - \frac{1}{\mathrm{CR}_w}. \tag{38}$$

Compression Bounds for Model Weights. The exact compression ratio depends on ρ_q^w, which can be measured directly from the encoded tensor. When only the zero-quotient block ratio ζ_w is available, Eq. 28 gives

$$\frac{1-\zeta_w}{N_B} \le \rho_q^w \le 1-\zeta_w. \tag{39}$$

Ignoring the negligible layer-base overhead, the average storage is therefore bounded by

$$\boxed{\begin{aligned} \bar{B}_{w,\min} &= U + k + \frac{1}{N_B} + (1-\zeta_w)\left(1 + \frac{b_q}{N_B}\right), \\ \bar{B}_{w,\max} &= U + k + \frac{1}{N_B} + (1-\zeta_w)(1+b_q). \end{aligned}} \tag{40}$$

Consequently,

$$\boxed{\frac{B}{\bar{B}_{w,\max}} \le \mathrm{CR}_w \le \frac{B}{\bar{B}_{w,\min}}.} \tag{41}$$

The lower compression-ratio bound corresponds to the conservative case in which every quotient in each active block is non-zero, whereas the upper bound corresponds to one non-zero quotient per active block.

KV Cache Storage. The KV cache compression follows the same blockwise encoding scheme but additionally retains the exponent vector of the first token in each KV page as the page-level base.

Consider G KV pages, each containing P tokens with D floating-point values per token. The total number of KV values is

$$N_{kv} = GPD. \tag{42}$$

The unchanged sign and mantissa fields require

$$S_{\mathrm{sm}}^{kv} = N_{kv}U \tag{43}$$

bits.

The exponent vector of the first token in each page is retained in its original E-bit representation. Since each page contains D base exponents, the base storage is

$$\begin{aligned} S_{\mathrm{ref}}^{kv} &= GDE \\ &= \frac{N_{kv}E}{P}. \end{aligned} \tag{44}$$

The remaining $P-1$ tokens are encoded as exponent residuals. The total number of residual values is therefore

$$\begin{aligned} N_r &= GD(P-1) \\ &= N_{kv}\frac{P-1}{P}. \end{aligned} \tag{45}$$

We concatenate residuals from consecutive KV pages into a contiguous stream before partitioning them into coding blocks, consistent with the encoding layout of `Loce`. Define

$$\alpha_P = \frac{P-1}{P}. \tag{46}$$

The remainder stream then requires

$$S_r^{kv} = N_{kv}\alpha_P k \tag{47}$$

bits, while the block-level bitmap contributes

$$S_h^{kv} = \frac{N_{kv}\alpha_P}{N_B} \tag{48}$$

bits.

Let ζ_{kv} denote the zero-quotient block ratio among KV residual blocks and ρ_q^{kv} denote the fraction of residual values with non-zero quotients. The intra-block bitmap and quotient payload require

$$S_g^{kv} = N_{kv}\alpha_P(1-\zeta_{kv}), \tag{49}$$

$$S_q^{kv} = N_{kv}\alpha_P\rho_q^{kv}b_q. \tag{50}$$

Combining all components gives

$$\begin{aligned} S_{kv} = N_{kv}U &+ \frac{N_{kv}E}{P} \\ &+ N_{kv}\alpha_P\left[k + \frac{1}{N_B} + (1-\zeta_{kv}) + \rho_q^{kv}b_q\right]. \end{aligned} \tag{51}$$

The average storage per KV value is therefore

$$\boxed{\bar{B}_{kv} = U + \frac{E}{P} + \frac{P-1}{P}\left[k + \frac{1}{N_B} + (1-\zeta_{kv}) + \rho_q^{kv} b_q\right].} \tag{52}$$

Accordingly, the KV cache compression ratio is

$$\boxed{\mathrm{CR}_{kv} = \frac{B}{U + \frac{E}{P} + \frac{P-1}{P}\left[k + \frac{1}{N_B} + (1-\zeta_{kv}) + \rho_q^{kv} b_q\right]}.} \tag{53}$$

The corresponding memory-saving ratio is

$$\mathrm{SR}_{kv} = 1 - \frac{\bar{B}_{kv}}{B}. \tag{54}$$

Compression Bounds for KV Cache. For KV residual blocks, the non-zero quotient ratio similarly satisfies

$$\frac{1-\zeta_{kv}}{N_B} \le \rho_q^{kv} \le 1 - \zeta_{kv}. \tag{55}$$

Substituting these bounds into Eq. 52 yields

$$\boxed{\begin{aligned} \bar{B}_{kv,\min} &= U + \frac{E}{P} + \frac{P-1}{P}\left[k + \frac{1}{N_B} + (1-\zeta_{kv})\left(1 + \frac{b_q}{N_B}\right)\right], \\ \bar{B}_{kv,\max} &= U + \frac{E}{P} + \frac{P-1}{P}\left[k + \frac{1}{N_B} + (1-\zeta_{kv})(1+b_q)\right]. \end{aligned}} \tag{56}$$

Hence,

$$\boxed{\frac{B}{\bar{B}_{kv,\max}} \le \mathrm{CR}_{kv} \le \frac{B}{\bar{B}_{kv,\min}}.} \tag{57}$$

These bounds include the page-level base overhead and require only the zero-quotient block ratio reported in our locality analysis.

BF16 Example. We use LLaMA3.3-70B Instruct as a concrete example. For BF16,

$$B = 16, \qquad E = 8, \qquad U = 8, \qquad k = 3, \qquad b_q = 6. \tag{58}$$

For model weights, Table 5 reports $\zeta_w = 90.3\%$. The fixed sign-and-mantissa, remainder, and block-bitmap storage is

$$8 + 3 + \frac{1}{256} = 11.0039 \quad \text{bits/value}. \tag{59}$$

Using Eq. 40, we obtain

$$11.103 \le \bar{B}_w \le 11.683 \quad \text{bits/value}, \tag{60}$$

which gives

$$\boxed{1.370\times \le \mathrm{CR}_w \le 1.441 \times .} \tag{61}$$

For KV cache tensors, the reported zero-quotient block ratio is $\zeta_{kv} = 97.1\%$. With $P = 16$, the fixed storage before active-block bitmap and quotient payloads is

$$\begin{aligned} B^{kv}_{\text{fixed}} &= 8 + \frac{8}{16} + \frac{15}{16}\left(3 + \frac{1}{256}\right) \\ &\approx 11.316 \quad \text{bits/value}. \end{aligned} \tag{62}$$

Equation 56 then gives

$$11.344 \le \bar{B}_{kv} \le 11.506 \quad \text{bits/value}, \tag{63}$$

corresponding to

$$\boxed{1.391\times \le \mathrm{CR}_{kv} \le 1.410 \times .} \tag{64}$$

The narrower KV cache interval results from its higher zero-quotient block ratio, which substantially limits the possible quotient payload even without explicitly measuring ρ_q^{kv}.

C Experimental and Implementation Details

C.1 Dataset Details and Reproducibility

We provide additional details on the datasets used in our experiments for reproducibility. For RULER (Hsieh et al., 2024), we use the QA tasks and generate long-context inputs with different context lengths by adjusting the number of distractor passages. The corresponding dataset and generation scripts are available at [1].

For LongBench (Bai et al., 2024), we follow the official benchmark configuration and evaluate on its 21 datasets covering QA, summarization, few-shot learning, synthetic tasks, and code completion. The dataset is publicly available at [2].

C.2 Generative AI Models Used for Evaluation

We evaluate `Loce` on a diverse set of generative AI models, including large language models (LLMs) and diffusion-based transformer models (DiTs). The detailed model configurations and corresponding resources are summarized as follows.

Large Language Models (LLMs).

- **OPT** (Zhang et al., 2022): A family of decoder-only pretrained language models released by Meta, covering model scales from 125M to 175B parameters. We use OPT-13B in our evaluation. The pretrained weights are available at [3].
- **Mistral-24B** (Jiang et al., 2023): A 24B-parameter decoder-only language model designed for efficient instruction following and long-context generation. We use the instruction-tuned version for evaluation. The model weights are available at [4].
- **Qwen3-32B** (Yang et al., 2025): A family of open large language models developed by Alibaba, including dense and mixture-of-experts variants with different model scales. We use the corresponding instruction-tuned models from the Qwen3 family. The pretrained weights are available at [5].
- **LLaMA-3.3-70B Instruct** (Grattafiori et al., 2024): A representative open-source LLM family developed by Meta. We evaluate the LLaMA-3.3-70B-Instruct model. The model weights are available at [6].

Diffusion-based Transformer Models (DiTs).

- **Wan2.2-TI2V-5B** (Wan et al., 2025): A video generation model based on diffusion transformers, designed for high-quality text-to-video and image-to-video generation. The model weights are available at [7].
- **Qwen-Image** (Wu et al., 2025): A 20B MMDiT image foundation model that achieves advances in complex text rendering. The model weights are available at [8].

C.3 Lossless Compression Baselines

We compare `Loce` with representative lossless compression methods under different precisions. The uncompressed model under each precision directly uses the original model weights. For BF16 models, we compare with general-purpose and LLM-oriented lossless compression approaches, including ZipNN, nvCOMP, NeuZip, DFloat11, and ZipServ. For FP8 models, we additionally include ECF8 as a floating-point compression baseline. Baseline details are provided below.

[1] https://huggingface.co/datasets/rbiswasfc/ruler
[2] https://huggingface.co/datasets/THUDM/LongBench
[3] https://huggingface.co/facebook/opt-13b
[4] https://huggingface.co/mistralai/Mistral-Small-24B-Instruct-2501
[5] https://huggingface.co/Qwen/Qwen3-32B
[6] https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct
[7] https://huggingface.co/Wan-AI/Wan2.2-TI2V-5B
[8] https://huggingface.co/Qwen/Qwen-Image

BF16 Compression Baselines.

- **Uncompressed (BF16).** The original BF16 model without compression.
- **ZipNN** (Hershcovitch et al., 2025). A lossless neural network compression framework that exploits weight redundancy through entropy coding. The implementation is available at [9].
- **nvCOMP** (Shanbhag et al., 2022). A GPU-accelerated compression library. We use its LZ4 compressor as a lossless compression baseline. The implementation is available at [10].
- **NeuZip** (Hao et al., 2024). A floating-point-aware lossless compression framework that exploits the low entropy of exponent bits in neural network weights and applies ANS encoding for compact storage. The implementation is available at [11].
- **DFloat11** (Zhang et al., 2026). A floating-point lossless compression method that exploits redundancy in floating-point representations and applies Huffman coding to achieve compact storage. The implementation is available at [12].
- **ZipServ** (Fan et al., 2026). An LLM serving-oriented lossless compression framework that exploits the concentration of weight values by using bitmap indexing for weights outside the seven frequent value intervals. The implementation is available at [13].

FP8 Compression Baselines.

- **Uncompressed (FP8).** The original FP8 model without compression.
- **ECF8** (Yang et al., 2026b). ECF8 is a lossless FP8 compression framework that exploits exponent concentration in floating-point weights and applies entropy-aware Huffman coding with GPU-optimized decoding. The implementation is available at [14].

FP4 Compression Baselines.

- **Uncompressed (FP4).** The original FP4 model without compression.

C.4 Software and Hardware

Hardware Configuration. We evaluate `Loce` on two GPU platforms with different memory capacities and architectures. ❶ *A100 GPU Server.* The first platform is equipped with 8× NVIDIA A100 SXM4 GPUs, each providing 80GB HBM memory. The server adopts a dual-socket Intel Xeon Platinum 8358P CPU configuration, with 64 physical cores and 128 threads in total. It is equipped with 1TB ECC DDR4-3200 system memory and 8TB NVMe SSD storage. The GPUs are interconnected through the high-bandwidth NVLink topology, enabling efficient multi-GPU communication. ❷ *RTX 5090 GPU Server.* The second platform is an ASUS ESC8000-E11 server equipped with 4× NVIDIA RTX 5090 GPUs. The system contains two Intel Xeon Platinum 8457C processors, providing 96 physical cores and 192 threads in total. It is configured with 512GB ECC DDR5-4800 memory and multiple NVMe SSDs for storage. The GPUs are connected through PCIe interconnects without NVLink support, representing a typical commodity GPU deployment environment.

Software Configuration. All experiments are conducted under Ubuntu Linux. Our experiments are conducted using PyTorch 2.7.1, CUDA 12.8, Transformers 4.56.0, and Diffusers 0.34.0. The kernels are implemented and optimized using CUDA programming interfaces.

D Supplementary Experimental Results

D.1 Analysis of Prefill and Decode KV Cache Footprint

Long-context inference introduces substantial KV cache memory overhead, as the cache size scales linearly with the number of cached tokens (Ye et al., 2025). The KV cache is generated from two

[9] https://github.com/zipnn/zipnn
[10] https://github.com/NVIDIA/nvcomp
[11] https://github.com/BorealisAI/neuzip
[12] https://github.com/LeanModels/DFloat11
[13] https://github.com/HPMLL/ZipServ_ASPLOS26
[14] https://github.com/zeyuyang8/ecf8

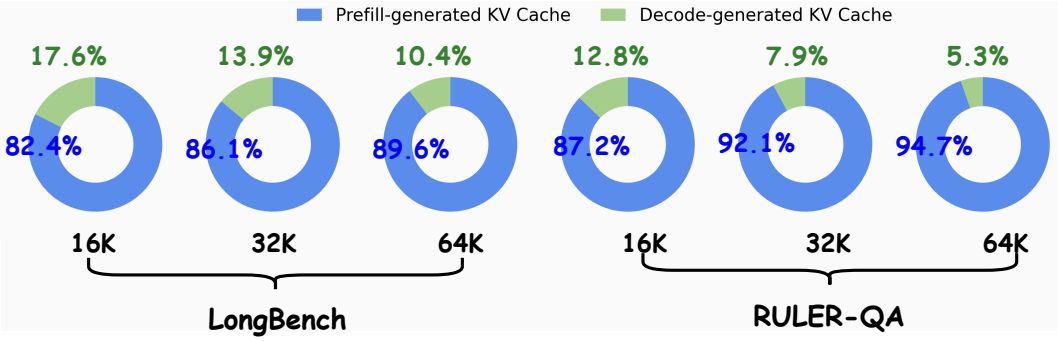

Figure 13: KV Cache Footprint Composition under Long-Context Inference.



Figure 14: The generated images from the original and Loce-compressed Qwen-Image models.

stages: the prefill stage, which processes input tokens in parallel, and the autoregressive decoding stage, which incrementally appends KV entries for generated tokens. Since long-context workloads usually contain much longer input contexts than generated responses, the prefill stage contributes the majority of the KV cache footprint. Recent KV cache optimization studies (Sun et al., 2024; Kwon et al., 2023) also highlight the memory pressure caused by long-context KV storage. To quantitatively validate this observation, we measure the KV cache composition under different context lengths on LongBench and RULER-QA. As shown in Figure 13, prefill-generated KV entries account for 82.4%-94.7% of the total KV footprint across evaluated context lengths.

This observation motivates our design choice of targeting the prefill-generated KV cache for compression. Since the majority of KV cache memory is created during the prefill stage, a single parallel encoding pass can compress most of the KV cache memory footprint, while avoiding repeated encoding operations for newly generated KV entries along the autoregressive decoding path.

D.2 Accuracy Evaluation

As a lossless compression framework, Loce preserves the exact model weights after decompression. The proposed Rice-based coding scheme is inherently lossless, as all exponent residuals are reversibly encoded and exactly reconstructed during decoding. Table 8 shows that Loce achieves identical accuracy and perplexity to the original BF16 model on Qwen3-32B across various benchmarks, including MMLU (Hendrycks et al., 2020), GPQA (Rein et al., 2023), WikiText-2 (Merity et al., 2016), and C4 (Dodge et al., 2021). We further evaluate Qwen-Image with compressed weights in Figure 14, where the generated images remain visually consistent with the original model.

Table 8: Comparison of accuracy and perplexity between BF16 and Loce-compressed models on different benchmarks. Loce compression results in absolutely no loss in accuracy or perplexity.

Model	Method	Accuracy		Perplexity	
		MMLU	GPQA	WikiText	C4
Qwen3-32B	BF16	83.61 ± 0.325	43.94 ± 1.660	7.60	10.77
	Loce (Ours)	**83.61 ± 0.325**	**43.94 ± 1.660**	**7.60**	**10.77**

Table 9: Comparison of Loce and BF16 on diffusion transformer models.

Model	Method	E2E Latency (s)	Memory (MB)	Memory ↓ (%)	Latency ↓ (%)
Wan2.2-TI2V-5B	BF16	231.44 ± 0.62	20480	0.0	0.0
	Loce(Weight)	**222.40 ± 0.101**	**17859**	**12.8**	**3.9**
Qwen-Image	BF16	36.11 ± 0.006	53674	0.0	0.0
	Loce(Weight)	**35.02 ± 0.57**	**45647**	**15.1**	**3.1**

D.3 Evaluation on Diffusion Transformers

Although Loce is designed for LLM inference, its exponent locality-based compression can be naturally extended to diffusion transformers (DiTs) with similar floating-point redundancy. We primarily apply Loce to DiT weight compression, since image generation models generally exhibit weaker KV cache locality than autoregressive language models, limiting the benefits of KV cache compression. We evaluate Loce on two representative generative models, Wan2.2-T2V-A5B and Qwen-Image. As shown in Table 9, Loce reduces the memory footprint by 17.8% and 15.1% while preserving generation quality. It also achieves 3.9% and 3.1% end-to-end latency improvements, respectively. The limited latency gains are attributed to the compute-bound nature of DiT workloads, where iterative transformer computation dominates over memory access.

D.4 Orthogonal to Low-Bit Floating-Point Quantization Techniques

Loce is orthogonal to low-bit floating-point quantization techniques and can be seamlessly integrated with low-precision KV cache representations. While FP8/FP4 quantization reduces memory footprint by lowering numerical precision, Loce further exploits the remaining exponent redundancy within the quantized representation to achieve additional lossless compression. Therefore, Loce can be applied on top of low-bit floating-point KV caches. For example, UltraQuant (Chakrabarti et al., 2026) is a recent low-bit floating-point KV cache quantization method that enables FP4 KV cache representation. As shown in Table 10, combining Loce with UltraQuant further improves the inference throughput of Qwen3-32B on four RTX 5090 GPUs under a 32K context length setting, achieving a 1.24× throughput improvement over the FP8 KV cache baseline.

Table 10: Performance of Loce combined with low-bit floating-point KV cache quantization methods.

KV format	Throughput (32K, batch 4)	Speedup
FP8 KV	51.67 tokens/s	1.00×
UltraQuant	59.93 tokens/s	1.16×
UltraQuant + Loce	**64.07 tokens/s**	**1.24**×

E Limitation Discussion and Future Work

Although Loce achieves efficient lossless compression for LLM weights and KV cache tensors, several directions remain open for further investigation.

❶ Loce currently focuses on compressing KV cache tensors generated during the prefill stage. This design choice is motivated by the observation that prefill-generated KV cache constitutes the majority of the total KV cache footprint in long-context inference, while continuously compressing newly generated KV entries during autoregressive decoding may introduce non-negligible encoding overhead. Future work could explore more adaptive online compression strategies, such as lightweight

incremental encoding or hardware-assisted KV cache compression, to further extend lossless compression benefits to dynamically generated KV entries.

❷ `Loce` is designed around the exponent locality observed in floating-point representations and supports BF16, FP8, and FP4 formats. Extending lossless compression to integer representations presents an interesting research direction, which may require identifying new forms of redundancy beyond exponent locality and developing data-type-aware compression schemes. Such extensions could further broaden the applicability of lossless compression.

❸ `Loce` can be extended to generative diffusion models, but its current design mainly targets weight compression, as the temporal locality observed in LLM KV cache does not directly transfer to diffusion workloads. Unlike text sequences, where contextual tokens often exhibit strong correlations, image representations may contain weaker sequential dependencies, making temporal locality-based compression less effective. Future work could investigate more fine-grained characteristics of diffusion KV caches, such as spatial, layer-wise, or denoising-step-wise redundancies, to develop specialized lossless compression strategies for multimodal generative models.

F USE OF LLM

We use large language models solely to refine wording and improve readability and fluency. No parts of the research methodology, experiments, data analysis, or conclusions were generated by LLMs. The authors take full responsibility for the final content of the paper. LLMs were not used for creating or fabricating research results.