InQuant: In-Place Mixed-Precision KV Cache Quantization via Saliency-Aware Neighbor-Slot Reuse

Zihan Chang[1,2,3] **Shuibing He**[1,2,3] * **Bo Zhou**[1,2] **Ping Chen**[1,4] **Siling Yang**[1]

[1]Zhejiang University [2]Zhejiang Lab

[3]Zhejiang Key Laboratory of Big Data Intelligent Computing

[4]Hangzhou High-Tech Zone (Binjiang) Institute of Blockchain and Data Security

Abstract

Key-value (KV) caches are essential for efficient Large Language Model (LLM) inference, but their memory footprint grows linearly with context length and batch size. Low-bit KV-cache quantization reduces this footprint, yet uniform quantization is vulnerable to high-magnitude outlier channels, while outlier-aware mixed-precision methods often introduce extra buffers, indexing, or channel permutation that weakens their system-level benefit.

This paper presents InQuant, an in-place mixed-precision KV-cache quantization method that preserves salient outlier channels without changing the physical 4-bit packed layout. The key observation is that selected low-saliency channels near outliers can tolerate small approximation error; InQuant therefore reuses their storage slots to hold the extra 4-bit nibble needed by 8-bit outlier values. To make this layout practical, InQuant combines sampling-based channel saliency estimation, saliency-aware neighbor-slot reuse, stride-based handling for adjacent outlier groups, and descriptor-guided marker recovery during dequantization. Across six LLMs and long-context workloads, InQuant reaches a fixed 4.0× physical KV-cache compression ratio, preserves accuracy close to strong mixed-precision baselines, and reduces quantization/dequantization latency by 2.2×–3.4× compared with representative outlier-aware methods. Our code is available at ISCS-ZJU/InQuant-open-source.

1 Introduction

Deep neural networks have demonstrated their effectiveness across a wide range of applications Chang et al. [2018], Brown et al. [2020], Chang et al. [2019]. Among them, large language models (LLMs) have emerged as a prominent class, supporting diverse applications Chang et al. [2026b], de Winter [2024], Chen et al. [2021], Chang et al. [2026a].

KV caching accelerates autoregressive decoding by storing reusable key–value states, but its memory footprint grows linearly with batch size and sequence length. As a result, KV-cache storage becomes a major bottleneck in long-context inference, making KV-cache compression an important optimization Sheng et al. [2023], Guo et al. [2023].

Existing KV cache compression approaches can be broadly categorized into two types, as illustrated in Figure 1. **(1) Uniform Quantization**. Traditional methods Kang et al. [2024] apply a uniform quantization scheme to the entire KV cache (Figure 1a). Although simple and hardware-friendly, this approach fails to distinguish outliers, which are activation values with large magnitudes. Quantizing them together with normal values under a shared scale significantly reduces the precision of outliers, resulting in severe inference accuracy loss He et al. [2024]. **(2) Outlier-Aware Mixed-Precision Methods**. Recent studies Lee et al. [2024], Liu et al. [2025], Yao et al. [2022], Zhao et al. [2024] observe that outliers in the KV cache tend to cluster within specific channels, where a channel typically consists either predominantly of outlier values (Outlier-channel) or of normal values (Normal-channel) (Figure 1b). Accordingly, these methods allocate higher bit-width quantization to outlier channels

*Corresponding author: heshuibing@zju.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

while applying lower bit-width quantization to normal ones. However, this fine-grained handling of outliers introduces additional data management and indexing overhead. A variant approach Liu et al. [2025] (Figure 1c) further performs channel permutation, rearranging channels containing outliers into contiguous blocks for block-wise mixed-precision quantization. Although this design enhances quantization locality, it inevitably incurs additional data movement overhead due to channel rearrangement, thereby limiting its practical efficiency, as discussed in §2.1.

We observe that normal values adjacent to outliers have limited influence on inference accuracy, allowing their storage to be reallocated for higher-precision representation of outliers (Figure 2). Motivated by this observation, we propose InQuant, a low-overhead in-place mixed-precision KV-cache quantization method that reuses low-saliency neighboring slots to provide higher-bitwidth representations for outlier channels, thereby achieving in-place mixed-precision quantization without explicit outlier separation or channel permutation, as shown in Figure 3. InQuant is designed around two constraints that are important for practical serving: the compressed KV cache should retain a regular packed memory footprint, and the outlier-preserving path should not require moving channels into separate buffers. To satisfy these constraints, InQuant uses a sampling-based channel saliency estimator, stores the extra nibble of each 8-bit outlier value in a selected low-saliency slot, and recovers the mixed layout with compact group descriptors during dequantization. The reported 4.0× compression ratio refers to the physical packed KV-cache footprint: InQuant achieves it by reusing selected low-saliency slots for the extra nibbles of 8-bit outliers, while approximating a small fraction of low-saliency entries. We report the corresponding overwritten ratio, recovery error, and quality impact in our analysis.

However, during KV cache compression, consecutive outlier channels may occur, meaning that a channel's neighboring channels are also outlier channels. In such cases, directly reallocating their neighboring storage would lead to loss of important information and degrade inference accuracy. To address this issue, InQuant introduces a stride-based in-place search strategy that scans non-adjacent low-importance channels to locate alternative reusable storage regions. This design preserves all critical data from adjacent outlier channels while still achieving efficient in-place mixed-precision quantization.

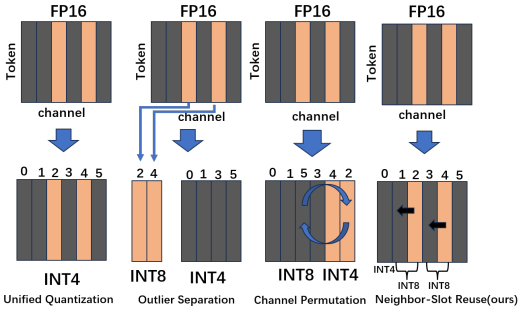

Figure 1: KV-cache quantization strategies. InQuant reuses neighboring low-impact storage for high-precision outlier representation.

Our contributions are summarized as follows. First, we propose an in-place mixed-precision KV-cache layout that represents salient channels at 8-bit precision within a fixed 4-bit physical footprint by reusing low-saliency slots. Second, we design a lightweight saliency-aware slot-selection and descriptor-guided recovery mechanism, including a stride-based extension for adjacent outlier groups. Third, we implement InQuant with fused CUDA kernels and evaluate it on six LLMs, standard accuracy benchmarks, and long-context serving workloads, showing that InQuant improves the compression–latency trade-off while retaining quality close to strong mixed-precision baselines.

2 Background and Motivation

2.1 Limitations of existing KV-cache quantization

Existing KV-cache quantization methods typically follow two directions. Uniform low-bit quantization applies the same precision to all KV entries, but large outliers can dominate the quantization scale and cause accuracy degradation. Outlier-aware mixed-precision methods preserve salient channels or tokens at higher precision, but usually require separate outlier handling, indexing, or channel permutation, which introduces additional data movement and layout-transformation overhead.

2.2 Opportunities

Observations. First, outlier channels are much more sensitive to precision loss than nearby normal channels. To validate this, we perform a zero-out sensitivity analysis on three LongBenchBai et al.

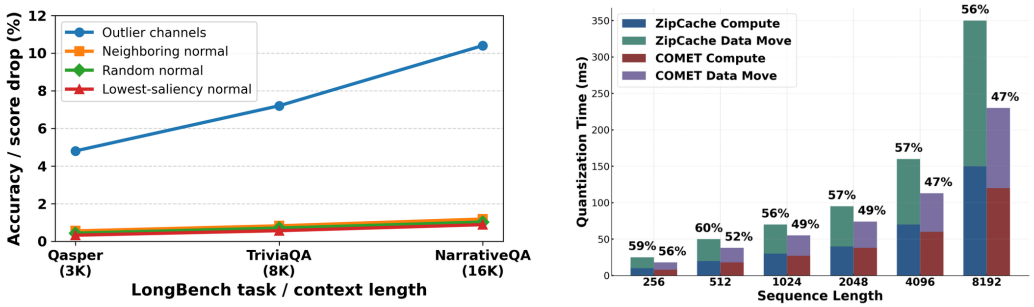

Figure 2: Motivation for InQuant. Left: zero-out sensitivity on LongBench shows that neighboring normal channels have much lower impact than outlier channels. Right: ZipCache and COMET spend 47%–60% of quantization time on data movement, motivating in-place neighbor-slot reuse.

[2024] tasks, Qasper, TriviaQA, and NarrativeQA, using Mistral-7B. We zero out the same 20% fraction of channels from four groups: outlier channels, neighboring normal channels, random normal channels, and globally lowest-saliency normal channels. As shown in Figure 2(left), removing outlier channels causes much larger quality degradation, while neighboring normal channels behave similarly to random and lowest-saliency normal channels. This suggests that neighboring low-impact slots can be reused to preserve high-precision outliers.

Second, existing mixed-precision KV-cache quantization methods incur substantial data-movement overhead. We profile representative outlier-aware and permutation-based methods, ZipCache and COMET, on Mistral-7B under varying input lengths. Figure 2(right) shows that data movement accounts for 47%–60% of total quantization time. These observations motivate InQuant to reuse low-impact neighboring storage for outliers while keeping the compressed layout in place.

These observations motivate InQuant: reuse low-impact neighboring storage for outliers and keep the layout in place to avoid data movement.

2.3 Challenge

Efficiently identifying outlier channels in the KV cache is the first challenge for in-place mixed-precision quantization. Outliers are sparse, making it too costly to scan the entire KV cache, while inaccurate estimation can significantly harm model accuracy. Therefore, we need a lightweight saliency estimator that avoids expensive full-tensor traversal for practical use.

A second challenge involves performing neighbor-slot reuse for both isolated and adjacent outlier channels. For isolated outliers, high-bit values must be safely placed into nearby low-saliency regions without corrupting important data. When outlier channels are adjacent, no immediate low-saliency neighbor exists, and inappropriate reuse strategy may overwrite meaningful values. Handling these cases requires carefully coordinated slot selection and dequantization strategies that preserve data integrity while maintaining efficiency and accuracy.

3 Design of InQuant

To address these challenges, we propose InQuant, an in-place mixed-precision quantization method for KV compression. The overall flowchart is shown in Figure 3. ① To avoid the high computational overhead of scanning the entire KV cache during outlier channel identification, we introduce a *sampling-based saliency estimation strategy*. ② To reduce the additional data movement overhead during quantization, we adopt an *in-place quantization scheme based on saliency-aware neighbor-slot reuse*, enabling high-bit out-

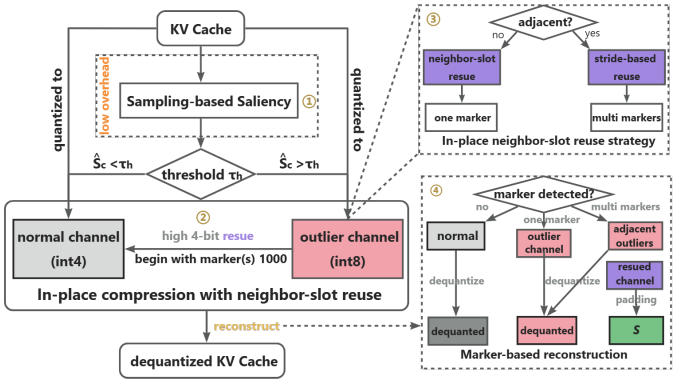

Figure 3: Overall workflow of InQuant.

lier values to be embedded directly within low-saliency regions. ③ To handle cases where outlier channels appear consecutively, we propose a *stride-based reusable-slot search strategy* that still enables stable in-place compression when there is no low-saliency channel directly next to them. ④ Because neighbor-slot reuse stores the extra outlier nibble in reused low-saliency slots, dequantization must identify and recover these displaced values. To keep this recovery lightweight, we design a *marker-based dequantization mechanism* and employ *sample-based padding* to restore reconstructed low-saliency channels with high fidelity.

3.1 Sampling-based Saliency-aware Channel Identification

Existing channel- or token-saliency estimators require full-cache statistics over tokens, heads, and batches Ge et al. [2023], Simonyan et al. [2013], Sundararajan et al. [2017], Zhang et al. [2023], He et al. [2024], which becomes costly for long-context autoregressive decoding. Based on the intra-channel distributional similarity of KV-cache activations Lee et al. [2024], Liu et al. [2025], InQuant estimates channel saliency by sampling a small fraction of token positions, e.g., one every 8 tokens, to approximate the L1-norm distribution. This reduces saliency-estimation overhead while maintaining ranking fidelity.

3.1.1 Sampling-based Saliency Score via L1-Norm.

To support fine-grained, in-place mixed-precision quantization of the KV cache, we design a sampling-based channel-wise saliency identification method grounded on the L1-norm of activation values.

We denote the KV cache tensor as:

$$\mathbf{X} \in \mathbb{R}^{B \times H \times L \times D}$$

where B is the batch size, H is the number of attention heads, L is the sequence length, and D is the channel dimension.

Instead of computing the full L1-norm across all tokens, we uniformly sample a subset of token indices $\mathcal{S} \subset \{1, \ldots, L\}$, typically with a sampling ratio $r = |\mathcal{S}|/L$. The sampling interval can be fixed (e.g., every 8th token) or adaptive based on cache growth. The saliency score of channel c is then estimated as:

$$S_c = \frac{L}{|\mathcal{S}|} \sum_{b=1}^{B} \sum_{h=1}^{H} \sum_{l \in \mathcal{S}} |\mathbf{X}_{b,h,l,c}|$$

where the scaling factor $\frac{L}{|\mathcal{S}|}$ compensates for the subsampling ratio to ensure unbiased estimation.

To enable relative comparison and thresholding, we apply min–max normalization across channels:

$$\hat{S}_c = \frac{S_c - \min_j S_j}{\max_j S_j - \min_j S_j}$$

This sampling-based saliency measure preserves the interpretability of traditional L1-based saliency while drastically reducing the computational and memory access overhead. Empirically, even a small sampling ratio (e.g., $r = 1/8$) yields channel rankings highly correlated (Spearman $\rho > 0.98$) with those obtained from full-cache L1 computation.

To further improve computational efficiency, we fuse the sampling-based L1-norm computation and min–max normalization into a single CUDA kernel. This kernel fusion strategy eliminates redundant GPU memory accesses and intermediate synchronization, leading to a more efficient in-place computation of saliency scores with minimal runtime overhead.

We assign bit-widths to each channel based on the normalized saliency score $\hat{S}_c$ as follows:

$$\text{bit-width}_c = \begin{cases} 8\text{-bit}, & \text{if } \hat{S}_c \geq \tau_h \\ 4\text{-bit}, & \text{if } \hat{S}_c < \tau_h \end{cases}$$

where τ_h denotes a saliency threshold used to distinguish high-importance and low-importance channels. Channels with higher normalized saliency are assigned higher precision (8-bit) to preserve representational fidelity, while less salient channels are quantized more aggressively (4-bit) to reduce memory footprint and computational cost.

3.2 In-place Compression with Saliency-Aware Neighbor-Slot Reuse

Bit-level layout. InQuant physically packs KV values at 4-bit granularity. Normal channels use one 4-bit slot per quantized value, while outlier channels selected for 8-bit precision are represented by two 4-bit nibbles. One nibble remains at the original outlier-channel position, and the other is stored in a reusable slot borrowed from a low-saliency channel selected by the neighbor-slot reuse rule. This layout allows InQuant to preserve high-precision outlier representation without allocating a separate outlier buffer or performing channel permutation. The physical compression ratio is therefore determined by the packed 4-bit KV array plus amortized metadata. The representation is not lossless for every low-saliency value: it deliberately trades a bounded amount of low-saliency information for preserving high-saliency outlier channels.

This in-place design intentionally approximates a small subset of low-saliency values. When a low-saliency channel is reused, the corresponding 4-bit slot is overwritten by the extra nibble of an outlier value, while the remaining slots are left unchanged. The overwritten low-saliency entries are reconstructed during dequantization using the sampling-based padding strategy. Since these channels are selected from low-saliency regions, the induced approximation has limited impact on attention outputs. We provide the detailed bit-level layout, metadata format, marker disambiguation rule, and reconstruction-error analysis in Appendix E.

Slot-selection and recovery invariants. For each quantization group, InQuant maintains a one-to-one mapping from every 8-bit outlier channel to one original slot and one reusable low-saliency slot. A reusable slot is assigned at most once, and the group descriptor records the outlier bitmap, reuse offset or stride, and bit-width type. Markers are only boundary hints at descriptor-valid positions; the descriptor is the source of layout correctness. Detailed pseudocode and decoding analysis are provided in Appendix D.

Quantization procedure. InQuant estimates saliency from sampled tokens, marks the top-k channels as 8-bit outliers, and scans channels in order. Isolated outliers borrow unused low-saliency neighbors, while adjacent outlier groups search reusable slots at stride distances from the group boundary. Rare cross-group fallbacks are recorded in the descriptor and included in the metadata overhead reported in Appendix E.

3.2.1 Quantization for Non-adjacent Outlier Channels.

For non-adjacent outliers, InQuant selects the immediate low-saliency neighbor as the reused slot and stores the second nibble of the 8-bit outlier there. As illustrated in Figure 4(a), the reused neighbor remains in the original packed layout, and its role is recorded in the group descriptor. We additionally place a lightweight boundary marker at descriptor-valid positions to accelerate recovery of reused regions during dequantization. The boundary marker records the start of the reused region and avoids per-token index buffers; it is never interpreted outside descriptor-valid positions.

3.2.2 Dequantization with Marker-Based Recovery.

As shown in Figure 4(b), dequantization restores the mixed KV layout by interpreting the group descriptor together with boundary markers. Markers are layout hints, not global sentinels: they are valid only at descriptor-specified channel boundaries. Reused low-saliency slots are reconstructed with sampled padding statistics S, while the two nibbles of each outlier are recombined into the original 8-bit code. This avoids per-token outlier indices and prevents ordinary 4-bit values from being misread as metadata; detailed decoding pseudocode and error analysis are in Appendix D and Appendix E.

3.2.3 Quantization for Adjacent Outlier Channels.

In real-world KV cache matrices, certain channels contain dense outlier values, and in some cases, such outlier channels appear adjacent to each other. Previous in-place mixed-precision quantization schemes rely on reusing the memory of low-saliency neighboring channels to store the high-bitwidth quantized data of outlier channels. However, when outlier channels are adjacent, this approach fails because no immediate low-saliency neighbor is available, resulting in either information loss or the inability to perform efficient compression.

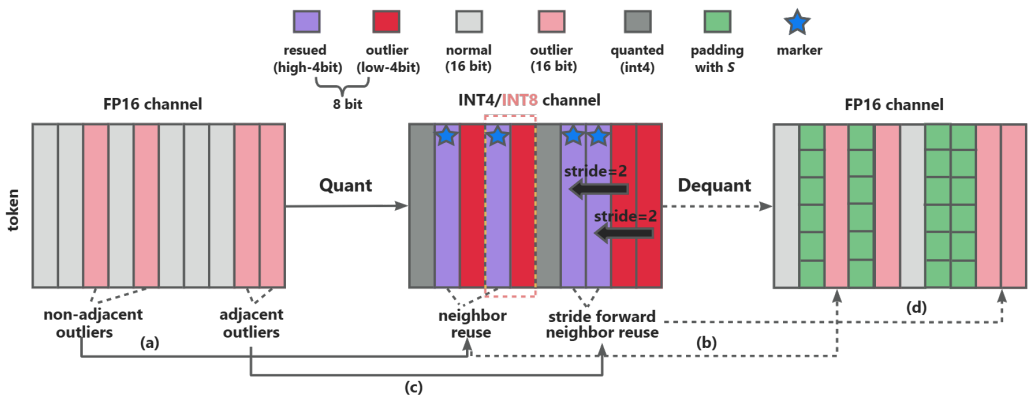

Figure 4: (a) In-place quantization for non-adjacent outlier channels using neighbor-slot reuse. (b) Marker-based dequantization for non-adjacent outlier channels, where reused low-saliency channels are restored using sampling-based padding from the subset S. (c) Stride-based reusable-slot search strategy for handling adjacent outlier channels. (d) Marker-based dequantization for adjacent outlier channels.

To address this challenge, we propose a *stride-based neighbor-slot reuse* strategy for in-place mixed-precision quantization. As illustrated in Figure 4(c), when multiple adjacent channels are detected as outliers, the algorithm searches for reusable low-saliency channels at a fixed stride distance (where the stride equals the number of adjacent outlier channels) away from the outlier group. This stride-based allocation allows densely clustered outliers to be stored in-place without separate outlier buffers. Moreover, each reused low-saliency region and its corresponding outlier block are jointly marked with a binary flag (e.g., `1000`), enabling efficient identification and precise recovery during dequantization while avoiding separate index tables.

3.2.4 Dequantization for Adjacent Outlier Channels.

For adjacent outlier groups, the descriptor records the same stride mapping used during quantization. The dequantizer follows this map to gather split nibbles from non-adjacent reusable slots and reconstructs overwritten low-saliency values with the same padding strategy. When stride boundaries cross alignment units, a lightweight remap aligns the affected segments before decoding; this adds less than 0.3% end-to-end overhead in our measurements.

4 Evaluation

4.1 Implementation Details

Models and datasets. To validate the efficiency of our approach, we conduct experiments with six open-source LLMs: GPT-2 1.3B Radford et al. [2019], LLaMA2-7B Touvron et al. [2023], LLaMA3-8B Dubey et al. [2024], Qwen2.5-7B Yang et al. [2025], Mistral-7B A. Q.Jiang [2023], and LLaMA2-13B Touvron et al. [2023]. We assess model performance on two established benchmarks: MMLU Hendrycks et al. [2021], a 57-task suite for knowledge evaluation, and GSM8K Cobbe et al. [2021], a benchmark designed for mathematical reasoning.

Quantization configuration. Unless otherwise specified, both key and value caches are quantized online with 4-bit group-wise quantization for normal channels and 8-bit quantization for the top 20% salient channels. The default saliency estimator samples one token every eight positions. Reported Q/DQ latency includes saliency estimation, packing, descriptor generation, marker-guided recovery, padding reconstruction, and adjacent-group remap. Additional reproducibility details are provided in Appendix A.

Baselines. We compare InQuant with four state-of-the-art approaches, (a) ZipCache He et al. [2024]: an accurate and efficient KV cache quantization method for LLMs, which performs mixed-precision compression on KV cache at channel-wise granularity by separately processing outliers and normal values. It applies high-bitwidth compression to high-saliency outliers while employing low-bitwidth compression for low-saliency values; (b) COMET Liu et al. [2025] adopts a block-wise compression approach. It first relocates outlier channels through channel permutation to consolidate them into

Table 1: Compact main results. MMLU and GSM8K are reported as average accuracy. CR denotes physical KV-cache compression ratio. Full category-level results are provided in Appendix B.

Model	Method	CR	MMLU Avg.	GSM8K Avg.	Q/DQ (ms)
Mistral-7B	FP16	1.0×	61.9	41.7	– / –
	ZipCache	2.47×	58.9	40.6	242.8 / 129.2
	COMET	2.31×	58.8	40.3	189.7 / 157.7
	InQuant	4.0×	58.4	40.3	78.4 / 58.4
LLaMA3-8B	FP16	1.0×	65.6	55.9	– / –
	ZipCache	2.47×	62.2	52.4	376.3 / 348.3
	COMET	2.31×	61.9	51.4	263.8 / 236.4
	InQuant	4.0×	61.8	52.4	96.5 / 67.4

dedicated blocks, then performs mixed-precision quantization separately for outlier and normal values at the block granularity; (c) GEAR Kang et al. [2024] achieves KV cache compression through a unified 4-bit quantization approach with error compensation mechanisms. (d) MiKVYang et al. [2024] is a mixed-precision quantization method that employs accumulated attention scores to estimate token saliency.

We provide additional discussion of other representative KV-cache quantization and serving systems, including KIVI, KVQuant, and QServe, in Appendix J. The main tables focus on baselines that expose the outlier-handling cost that InQuant targets: separate salient-channel processing, token/channel saliency tracking, and channel permutation. KIVI, KVQuant, and QServe are complementary low-bit formats or full serving stacks; we therefore discuss them separately and avoid presenting them as controlled baselines for the in-place layout itself. To avoid overstating the scope, our main claim is not that InQuant dominates all KV-cache quantization formats. Rather, InQuant improves the system cost of outlier-aware mixed-precision KV compression by preserving salient channels inside an in-place physical layout. The controlled 4-bit-footprint variants in Table 2 isolate this layout contribution, while Appendix J explains how low-bit formats such as KIVI and KVQuant are orthogonal to the proposed layout.

Evaluation protocol and hardware. We use deterministic decoding for MMLU and GSM8K, report latency as the mean of five measured runs after two warm-up runs, and evaluate all methods under the same sequence length, memory budget, and serving configuration. Long-context experiments use batch size 8, 1K–32K input tokens, and 128 generated tokens. Experiments run on an Intel Xeon 96-core CPU and a single NVIDIA A100 80GB GPU.

4.2 General Results

4.2.1 Evaluation on MMLU and GSM8K.

We evaluate InQuant on MMLU and GSM8K to measure the accuracy–efficiency trade-off of KV-cache quantization. Table 1 reports representative results on Mistral-7B and LLaMA3-8B, including average accuracy, physical compression ratio, and quantization/dequantization latency. The complete category-level MMLU results, GSM8K metric breakdown, additional models, and all baselines are provided in Appendix B.

As shown in Table 1, InQuant achieves a fixed 4.0× physical KV-cache compression ratio while maintaining accuracy close to strong mixed-precision baselines. Compared with ZipCache and COMET, InQuant substantially reduces quantization/dequantization latency by avoiding explicit outlier separation, channel permutation, and costly data movement. For example, on Mistral-7B, InQuant reduces quantization latency to 78.4 ms, compared with 242.8 ms for ZipCache and 189.7 ms for COMET; on LLaMA3-8B, it reduces quantization latency to 96.5 ms, compared with 376.3 ms and 263.8 ms, respectively. These results show that InQuant provides a favorable accuracy–compression–latency trade-off under the same saliency-ratio setting.

Clarifying compression ratio. We report the *physical KV-cache compression ratio*, i.e., the actual packed memory footprint after metadata amortization, rather than a lossless information-preserving mixed-precision storage ratio. InQuant attains a 4.0× physical footprint because the extra nibble of each selected 8-bit outlier value is stored in a reused low-saliency slot, whose original value is

Table 2: Fairness analysis under comparable physical KV-cache footprints on Mistral-7B. MMLU Avg. is averaged over STEM, Humanities, Social Sciences, and Other. LongBench Avg. is averaged over Qasper, HotpotQA, MultiFieldQA, GovReport, and NarrativeQA. All methods use the same 20% saliency ratio when applicable.

Method	Physical CR	Overwritten Ratio	MMLU Avg.	LongBench Avg.	Quant. (ms)	Dequant. (ms)
Uniform INT4	4.0×	0%	52.8	35.1	56.4	41.8
INT4 + Low-saliency Padding	4.0×	20%	53.6	35.4	64.8	50.3
Random Neighbor Reuse	4.0×	20%	56.1	36.8	70.5	54.6
InQuant w/o Padding	4.0×	20%	57.6	37.7	76.9	52.1
InQuant Full	4.0×	20%	58.4	38.3	78.4	58.4
ZipCache	2.47×	0%	58.9	38.5	242.8	129.2

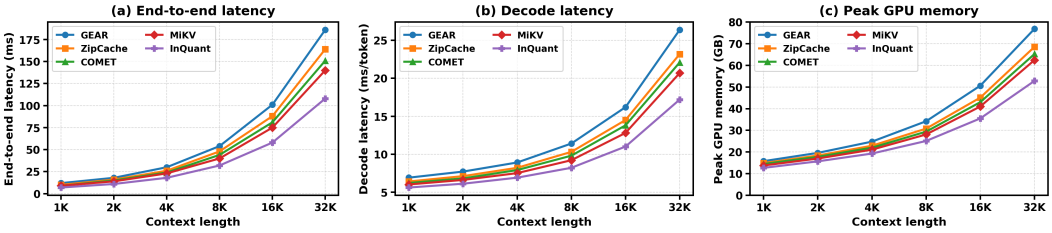

Figure 5: Long-context end-to-end inference on LLaMA3-8B. We vary the context length from 1K to 32K and report end-to-end latency, decoding latency, and peak GPU memory. InQuant consistently reduces latency and memory usage, with larger gains as the KV cache grows.

approximated and recovered during dequantization. This metric is the relevant memory-capacity metric for serving, but it should be interpreted together with the overwritten ratio and quality impact. To make the comparison transparent, Table 2 compares InQuant with controlled 4-bit-footprint variants, while Appendix E reports metadata overhead, marker collision rate, padding reconstruction error, and remap overhead.

Fairness analysis. Since InQuant achieves a fixed 4-bit physical footprint by reusing low-saliency neighboring slots for the extra nibble of high-precision outlier values, we further examine whether its gains simply come from discarding low-saliency information. Table 2 compares InQuant with several controlled variants under comparable physical KV-cache footprints. Uniform INT4 suffers from noticeable quality degradation because it does not preserve outlier channels. INT4 with low-saliency padding provides only marginal improvement, showing that reconstructing low-saliency values alone is insufficient. Random neighbor reuse improves over uniform INT4 by preserving some outlier information, but still underperforms InQuant due to non-saliency-aware overwriting. Removing the padding-based recovery from InQuant also degrades quality, confirming the importance of reconstructing reused low-saliency slots. Full InQuant achieves quality close to ZipCache while providing a fixed 4-bit footprint and much lower quantization/dequantization latency, demonstrating that its benefit comes from saliency-aware neighbor-slot reuse rather than merely dropping less important KV-cache entries.

4.2.2 Long-context End-to-end Evaluation.

We evaluate InQuant under long-context inference to test whether its efficiency advantage holds as the KV cache grows. Unless otherwise specified, we use LLaMA3-8B with batch size 8 on a single NVIDIA A100 80GB GPU, vary the input context length from 1K to 32K, and generate 128 tokens per request. All methods are evaluated under the same serving configuration, memory budget, and quantization setting. We report average latency and peak GPU memory over five runs after two warm-up runs, and compare against GEAR, ZipCache, COMET, and MiKV.

Table 3: Long-context quality and retrieval recall. LB-5 averages five LongBench tasks; Needle and Passkey report 32K retrieval accuracy on LLaMA3-8B. Higher is better.

Method	LB-5 Avg.		Needle	Passkey
	LLaMA3	Mistral	32K	32K
FP16	40.9	39.2	98.8	98.4
GEAR	37.1	35.8	90.5	89.8
ZipCache	40.0	38.5	97.4	96.8
InQuant	39.8	38.3	97.1	96.5

As shown in Figure 5, InQuant consistently achieves lower end-to-end latency, decoding latency, and peak GPU memory usage across all context lengths. The advantage becomes more pronounced at longer contexts, where KV-cache quantization, dequantization, and data movement costs are

Table 4: RULER v1 and AIME 2026 evaluation on Qwen2.5-7B-Instruct. RULER averages 13 tasks with five samples per task. AIME uses 30 problems with greedy decoding and an 8192-token output limit. Storage includes persistent KV-cache and decode state at 32K. We compare InQuant with SnapKV Li et al. [2024], K-Norm Devoto et al. [2024], and ZipCache He et al. [2024].

	RULER v1 (↑)			AIME 2026		Memory Efficiency	
Method	8K	16K	32K	Acc. (%) ↑	Cap Stops	Storage (MiB) ↓	Reduction (%) ↑
BF16	90.90	92.56	89.72	10.00	1/30	1784.78	0.00
InQuant	**90.90**	**92.10**	**88.03**	**6.67**	3/30	**386.77**	**78.21**
SnapKV	82.13	77.33	78.85	6.67	4/30	892.72	49.94
Knorm	32.97	35.44	31.59	0.00	3/30	892.72	49.91
ZipCache	70.26	61.87	37.90	0.00	1/30	405.63	77.20

All experiments use BF16 model weights, Transformers 4.53.3, and batch size 1. InQuant uses K4/V2; SnapKV and Knorm retain 50% of prefill tokens, while ZipCache uses its 4/2-bit preset with 40% unimportant tokens. Storage reports per-request medians; reduction averages paired reductions relative to BF16. Compression methods use different cache budgets. Bold indicates the best compressed-method results. Cap Stops counts generations reaching the 8192-token limit.

amplified. Compared with ZipCache and COMET, InQuant avoids explicit outlier separation and channel permutation; compared with MiKV, it reduces saliency-tracking overhead through sampling-based channel saliency and in-place neighbor-slot reuse. These results show that marker-based recovery and stride-based adjacent-outlier handling introduce limited overhead while preserving an efficient compressed layout.

Since InQuant reuses low-saliency storage to preserve high-saliency outliers, we further evaluate whether this approximation harms long-context quality and retrieval recall. Table 3 reports both the LongBench LB-5 average and retrieval-style accuracy at 32K context length. InQuant remains close to ZipCache on LongBench and only slightly trails it on Needle-in-a-HaystackNelson et al. [2024] and RULER passkey retrievalHsieh et al. [2024], while providing lower latency and memory usage as shown in Figure 5. These results indicate that the efficiency gains of InQuant do not come at the cost of long-context recall quality. Detailed task-level LongBench results and additional retrieval-style evaluations are provided in Appendix H.

Additional Long-Context and Reasoning Evaluation.

We further evaluate InQuant on the full RULER v1 benchmark and AIME 2026 using Qwen2.5-7B-Instruct. As shown in Table 4, InQuant consistently achieves the highest RULER scores among compressed methods across 8K–32K contexts, matching BF16 at 8K and trailing by only 1.69 points at 32K (88.03 vs. 89.72), while reducing persistent KV-cache and decode-state storage by 78.21%. Notably, at 32K, InQuant outperforms ZipCache by 50.13 points (88.03 vs. 37.90) with a slightly smaller memory footprint (386.77 vs. 405.63 MiB). On AIME 2026, InQuant achieves 2/30 correct answers, compared with 3/30 for BF16, and matches the best compressed baseline. These results further demonstrate InQuant's favorable trade-off between long-context accuracy and memory efficiency.

4.3 Ablation Study

4.3.1 Ablation on Sampling-based Saliency Estimation

We further evaluate the reliability and efficiency of the proposed sampling-based saliency estimator. The experiment is conducted on Mistral-7B using the MMLU benchmark under the same quantization setting as the main experiments, where the top 20% salient channels are quantized to 8 bits and the remaining channels are quantized to 4 bits. We vary the sampling ratio from $1/2$ to $1/16$ and use full-cache L1-norm saliency as the reference. Figure 6 evaluates different sampling ratios for saliency estimation. A ratio of $1/8$ achieves nearly the same saliency ranking and MMLU accuracy as denser sampling, while substantially reducing estimation latency. We therefore use $1/8$ as the default setting.

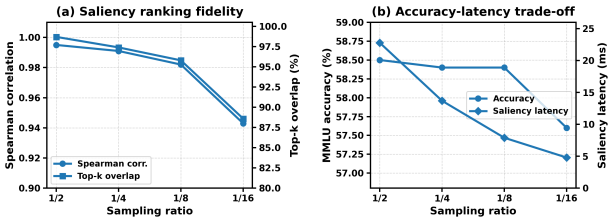

Figure 6: Sampling-based saliency ablation on Mistral-7B/MMLU. A sampling ratio of $1/8$ preserves saliency ranking and accuracy while reducing estimation latency.

4.3.2 Ablation on Adjacent Outlier Handling

We evaluate the necessity of the proposed adjacent outlier handling mechanism under the same quantization configuration as the main experiments, where the top 20% salient channels are quantized to 8 bits and the remaining channels are quantized to 4 bits.

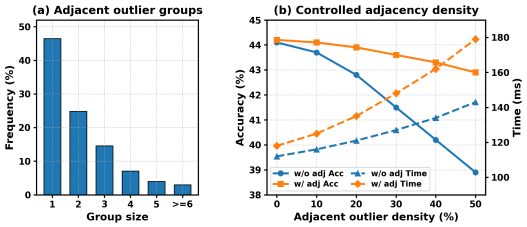

Figure 7: Adjacent outlier ablation. Full InQuant preserves accuracy under higher adjacent-outlier density.

Figure 7(a) reports the distribution of adjacent outlier group sizes measured on Mistral-7B inputs with an 8K context length. Although isolated outliers are the most common case, multi-channel adjacent outlier groups also appear frequently, showing that adjacency is not a rare corner case. This motivates explicitly handling consecutive outlier channels instead of relying only on immediate neighboring-slot reuse.

Figure 7(b) further evaluates accuracy and quantization/dequantization latency under controlled adjacent-outlier density on Mistral-7B and MMLU. As adjacent-outlier density increases, InQuant without adjacent-outlier handling suffers larger accuracy degradation. In contrast, full InQuant with stride-based reusable-slot search maintains substantially higher accuracy with only moderate latency overhead, confirming the robustness of the proposed adjacent-outlier handling mechanism.

5 Conclusion

In this work, we present InQuant, an in-place mixed-precision KV-cache quantization method that preserves salient outlier channels by reusing selected low-saliency neighboring slots. By avoiding explicit outlier buffers and channel permutation, InQuant reduces data movement and quantization/dequantization latency while maintaining a fixed 4-bit physical footprint. We further introduce stride-based reusable-slot search to handle adjacent outlier channels. Experiments on standard and long-context benchmarks show that InQuant achieves a favorable compression–latency–accuracy trade-off, with quality close to strong mixed-precision baselines.

Limitations

InQuant assumes that a small fraction of low-saliency KV entries can be approximated with limited effect on attention, and it requires descriptor-aware quantization/dequantization kernels. Our experiments validate this trade-off on open-source models up to 13B parameters and 32K context, but larger models, additional GPU architectures, and combinations with sub-4-bit formats such as KIVI or KVQuant remain future work.

Acknowledgments and Disclosure of Funding

This work was supported by the Science and Technology Program of Zhejiang Province (2026SDXT012), Sugon-ZJU-Research Projects (2025-KYY-511120-0021), Major Projects of Zhejiang Province (LD24F020012), National Science Foundation of China (U25A20423), Hangzhou Chengxi Sci-Tech Innovation Corridor Special Development Fund (K20241957).

The authors declare no competing financial interests.

References

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

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137, 2024.

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

Zihan Chang, Yang Zhang, and Wenbo Chen. Effective adam-optimized lstm neural network for electricity price forecasting. In *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, pages 245–248. IEEE, 2018.

Zihan Chang, Yang Zhang, and Wenbo Chen. Electricity price prediction based on hybrid model of adam optimized lstm neural network and wavelet transform. *Energy*, 187:115804, 2019.

Zihan Chang, Shuibing He, Bo Zhou, Sheng Xiao, Siling Yang, Rui Wang, and Zhe Pan. Spiderflow: Efficient topology-aware scheduling for llm training across decentralized gpu clusters. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13603–13615, 2026a.

Zihan Chang, Sheng Xiao, Shuibing He, Xuechen Zhang, Siling Yang, Zhenxin Li, Zhe Pan, and Weijian Chen. Frenzy: A memory-aware serverless llm training system for heterogeneous gpu clusters. In *European Conference on Parallel Processing*, pages 91–105. Springer, 2026b.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Joost CF de Winter. Can chatgpt pass high school exams on english language comprehension? *International Journal of Artificial Intelligence in Education*, 34(3):915–930, 2024.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for kv cache compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18476–18499, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.

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

Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–15, 2023.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

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.

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

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, 2024.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Lian Liu, Long Cheng, Haimeng Ren, Zhaohui Xu, Yudong Pan, Mengdi Wang, Xiaowei Li, Yinhe Han, and Ying Wang. Comet: Towards practical w4a4kv4 llms serving. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 131–146, 2025.

Elliot Nelson, Georgios Kollias, Payel Das, Subhajit Chaudhury, and Soham Dan. Needle in the haystack for memory based large language models. *arXiv preprint arXiv:2407.01437*, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

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

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, et al. Qwen2. 5-1m technical report. *arXiv preprint arXiv:2501.15383*, 2025.

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

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in neural information processing systems*, 35:27168–27183, 2022.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

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

Table 5: Serving-level breakdown on LLaMA3-8B at 32K context length with batch size 8. InQuant reduces decode latency, Q/DQ overhead, and peak HBM usage, leading to higher generated-token throughput.

Method	Prefill (s)	Decode (ms/token)	Q/DQ (%)	Peak HBM (GB)	Throughput (tok/s)
GEAR	180.3	28.6	11.6	78.2	5.57
ZipCache	167.8	26.4	21.2	72.6	5.98
MiKV	155.4	24.8	21.7	69.8	6.46
COMET	143.6	22.8	16.8	63.5	6.99
InQuant	129.5	20.7	7.0	56.1	7.75

Appendix

Broader Impact

InQuant is an infrastructure technique for improving the efficiency of long-context LLM inference by reducing KV-cache memory usage and quantization/dequantization overhead. Its potential positive impact is to lower the hardware and energy cost of serving long-context models, making such systems more accessible to researchers and developers with limited GPU resources.

However, more efficient inference may also increase the scale at which LLMs are deployed, including potentially harmful uses such as misinformation generation, spam, privacy-sensitive analysis, or low-quality automated content. InQuant does not introduce new model capabilities, but it can make existing capabilities cheaper to run. Deployments should therefore be paired with appropriate safety, privacy, and monitoring mechanisms.

InQuant also introduces a controlled approximation by reusing low-saliency KV-cache slots to preserve high-saliency outlier channels. Although our experiments show limited quality degradation on standard and long-context benchmarks, users should validate the method under their own models, context lengths, and application workloads, especially in high-stakes settings.

A Additional Reproducibility Details

InQuant uses a fixed saliency ratio of 20% in the main experiments unless otherwise specified. The default saliency sampling ratio is $1/8$, and the sampling ablation in Figure 6 studies the sensitivity to this choice. Quantization and dequantization latency include all online layout operations introduced by InQuant, including saliency estimation, descriptor construction, marker-guided recovery, padding reconstruction, and cross-block remap for adjacent outlier groups. For long-context experiments, all methods use batch size 8, context lengths from 1K to 32K, and 128 generated tokens per request; latency and memory are averaged over five measured runs after two warm-up runs. All accuracy evaluations use deterministic decoding with temperature 0. We evaluate baselines under the same context length, memory budget, and saliency ratio when their method exposes a comparable ratio. For methods without a saliency-ratio control, we use the configuration recommended by the original method and report the resulting physical KV-cache footprint.

A.1 Serving-level Breakdown

To further quantify the system-level effect of reducing online KV-cache quantization and dequantization overhead, we report a serving-level breakdown under the longest context setting used in Figure 5. All methods are evaluated on LLaMA3-8B with batch size 8, 32K input tokens, and 128 generated tokens on a single NVIDIA A100 80GB GPU. Throughput is computed as the number of generated tokens divided by the measured end-to-end latency. The Q/DQ percentage includes online saliency estimation, packing, descriptor generation, marker-guided recovery, padding reconstruction, and adjacent-group remap when applicable.

B Detailed MMLU and GSM8K Results

Table 6 provides the complete accuracy and efficiency results corresponding to the compact main results in Table 1. For MMLU, we report category-level accuracy over STEM, Humanities, Social Sciences, and Other. For GSM8K, we report both normalized accuracy and average accuracy. We

include all evaluated baselines under the same saliency-ratio setting when applicable, together with the physical KV-cache compression ratio and quantization/dequantization latency. These detailed results show that InQuant consistently achieves a fixed 4-bit physical footprint, substantially lower quantization/dequantization latency, and accuracy close to strong mixed-precision baselines across models and benchmarks.

Table 6: Detailed performance comparisons on MMLU and GSM8K. MMLU reports category-level accuracy over STEM, Humanities, Social Sciences, and Other; GSM8K reports normalized accuracy and average accuracy. CR denotes physical KV-cache compression ratio, and Q/DQ denote quantization/dequantization latency.

Model	Method	Saliency Rat.	Comp. Rat.	STEM	Human.	Metric-1	Metric-2	Quant. time (ms)	Dequant. time (ms)
(a) MMLU Benchmark									
Mistral-7B	FP16	100%	1.0×	40.2	70.9	65.0	71.4	–	–
Mistral-7B	GEAR	100%	3.0×	37.1	65.2	56.2	61.7	179.34	158.27
Mistral-7B	MiKV	20%	2.47×	38.8	68.1	61.3	66.0	249.36	134.72
Mistral-7B	ZipCache	20%	2.47×	39.1	68.5	61.7	66.4	242.82	129.19
Mistral-7B	COMET	20%	2.31×	38.8	68.9	61.3	66.1	189.73	157.66
Mistral-7B	InQuant	20%	4.0×	38.1	68.3	61.2	66.1	78.42	58.39
GPT-2 1.3B	FP16	100%	1.0×	27.8	33.4	35.2	34.6	–	–
GPT-2 1.3B	GEAR	100%	3.0×	25.3	29.4	29.7	28.3	165.44	152.90
GPT-2 1.3B	MiKV	20%	2.47×	27.4	32.5	31.7	32.5	281.46	260.37
GPT-2 1.3B	ZipCache	20%	2.47×	27.7	32.8	32.0	32.8	274.19	254.83
GPT-2 1.3B	COMET	20%	2.31×	27.2	32.5	31.3	32.1	205.71	179.22
GPT-2 1.3B	InQuant	20%	4.0×	27.2	32.4	31.6	32.1	82.56	51.93
Llama3-8B	FP16	100%	1.0×	52.5	65.5	76.9	67.3	–	–
Llama3-8B	GEAR	100%	3.0×	42.8	53.6	69.4	66.3	194.27	174.88
Llama3-8B	MiKV	20%	2.47×	49.4	59.7	71.6	66.5	384.91	355.72
Llama3-8B	ZipCache	20%	2.47×	49.8	60.1	72.0	66.8	376.34	348.29
Llama3-8B	COMET	20%	2.31×	49.9	58.7	72.5	66.3	263.82	236.41
Llama3-8B	InQuant	20%	4.0×	49.7	58.7	72.4	66.3	96.52	67.38
(b) GSM8K Benchmark									
Mistral-7B	FP16	100%	1.0×	–	–	41.68	41.62	–	–
Mistral-7B	GEAR	100%	3.0×	–	–	37.31	37.28	144.65	137.52
Mistral-7B	MiKV	20%	2.47×	–	–	40.35	40.29	242.83	227.16
Mistral-7B	ZipCache	20%	2.47×	–	–	40.62	40.57	236.49	221.34
Mistral-7B	COMET	20%	2.31×	–	–	40.35	40.32	176.82	154.33
Mistral-7B	InQuant	20%	4.0×	–	–	40.29	40.28	74.43	52.77
GPT-2 1.3B	FP16	100%	1.0×	–	–	13.54	13.52	–	–
GPT-2 1.3B	GEAR	100%	3.0×	–	–	11.52	11.48	132.44	124.61
GPT-2 1.3B	MiKV	20%	2.47×	–	–	12.54	12.49	276.42	254.61
GPT-2 1.3B	ZipCache	20%	2.47×	–	–	12.73	12.68	269.91	248.55
GPT-2 1.3B	COMET	20%	2.31×	–	–	12.45	12.44	189.34	163.22
GPT-2 1.3B	InQuant	20%	4.0×	–	–	12.45	12.44	78.27	54.28
Llama3-8B	FP16	100%	1.0×	–	–	55.89	55.88	–	–
Llama3-8B	GEAR	100%	3.0×	–	–	48.47	48.43	162.83	153.55
Llama3-8B	MiKV	20%	2.47×	–	–	52.18	52.14	321.64	304.28
Llama3-8B	ZipCache	20%	2.47×	–	–	52.45	52.43	313.39	297.41
Llama3-8B	COMET	20%	2.31×	–	–	51.41	51.36	225.72	198.66
Llama3-8B	InQuant	20%	4.0×	–	–	52.38	52.33	81.25	65.42

Note: Comp. Rat. = compression ratio, Human. = Humanities, Soc.Sci. = Social Sciences. For MMLU, Metric-1/2 = Soc.Sci./Other; for GSM8K, Metric-1/2 = Acc Norm./Avg Acc.

C Statistics of Adjacent Outlier Channels

We provide additional statistics to verify that adjacent outlier channels are not rare corner cases in KV-cache activations. This analysis supports the need for the stride-based adjacent outlier handling mechanism introduced in Section 3.2. Unless otherwise specified, we collect the statistics using MMLU inputs with an 8K context length. We use the same saliency definition and quantization configuration as in the main experiments: channel saliency is estimated by the proposed sampling-based L1 score, and the top 20% salient channels are treated as outlier channels. The remaining channels are treated as normal channels.

Definition of adjacent outlier groups. Given the binary outlier-channel indicator along the channel dimension, we define an adjacent outlier group as a maximal consecutive segment of outlier channels. For example, if channels c, $c+1$, and $c+2$ are all selected as outlier channels, and channels $c-1$

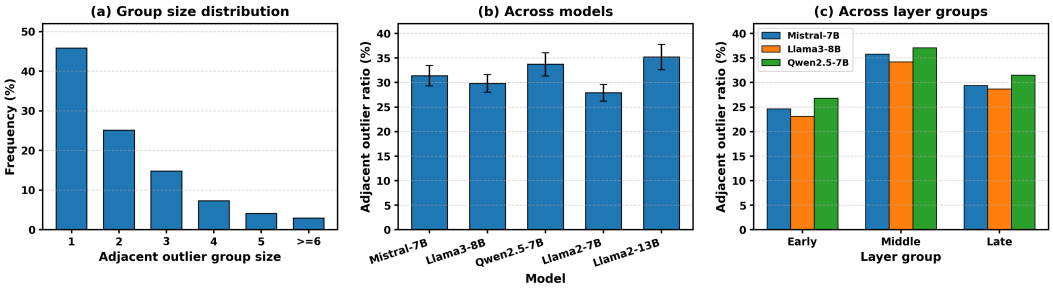

Figure 8: Statistics of adjacent outlier channels. Outlier channels are selected as the top 20% salient channels according to the sampling-based L1 saliency score. Unless otherwise specified, statistics are collected using MMLU inputs with an 8K context length. (a) Distribution of adjacent outlier group sizes averaged over the evaluated models. A group size of 1 denotes an isolated outlier channel, while group size ≥ 2 denotes consecutive outlier channels. (b) Adjacent outlier ratio across different models, where the error bars denote the standard deviation across layers. (c) Adjacent outlier ratio across early, middle, and late layer groups for representative models. The results show that adjacent outliers occur consistently across models and layers, motivating the proposed stride-based adjacent outlier handling mechanism.

and $c+3$ are normal channels, then these three channels form one adjacent outlier group with group size 3. A group size of 1 corresponds to an isolated outlier channel, while a group size larger than 1 indicates the presence of adjacent outlier channels. We further define the adjacent outlier ratio as

$$\text{Adjacent Ratio} = \frac{\{\text{outlier channels belonging to groups with size} \geq 2\}}{\{\text{all outlier channels}\}}.$$

This metric measures the fraction of outlier channels that cannot be handled by directly borrowing storage from immediate neighboring normal channels.

Figure 8(a) reports the distribution of adjacent outlier group sizes averaged over the evaluated models. Although isolated outlier channels remain the most frequent case, a non-negligible fraction of outlier channels appear in consecutive groups with size larger than one. In particular, groups of size 2 and 3 account for a substantial portion of all outlier groups, and longer groups also appear occasionally. This observation indicates that adjacent outliers are not merely pathological cases, but a recurring pattern in KV-cache activations.

Figure 8(b) further compares the adjacent outlier ratio across different model architectures, including Mistral-7B, Llama3-8B, Qwen2.5-7B, Llama2-7B, and Llama2-13B. The adjacent outlier ratio remains non-trivial for all tested models, showing that this phenomenon is not specific to a single architecture. The error bars denote the standard deviation across layers, suggesting that although the exact ratio varies by layer, adjacent outlier channels consistently appear throughout the model.

Figure 8(c) analyzes the layer-wise pattern by grouping transformer layers into early, middle, and late stages according to relative depth. For a model with N transformer layers, we define early layers as $[0, \lfloor N/3 \rfloor - 1]$, middle layers as $[\lfloor N/3 \rfloor, \lfloor 2N/3 \rfloor - 1]$, and late layers as $[\lfloor 2N/3 \rfloor, N-1]$. Accordingly, Mistral-7B and Llama3-8B are divided into layers 0–9, 10–20, and 21–31, while Qwen2.5-7B is divided into layers 0–8, 9–17, and 18–27. Across representative models, middle layers tend to exhibit a higher adjacent outlier ratio than early and late layers. This suggests that adjacent outliers are not uniformly distributed, and some layers are more likely to contain clustered high-saliency channels. Such layer-dependent clustering makes it insufficient to assume that every outlier channel has an immediately available low-saliency neighbor.

Overall, these statistics confirm that adjacent outlier channels are common enough to affect practical in-place mixed-precision KV-cache quantization. Without explicitly handling this case, neighbor-slot reuse may overwrite important adjacent outlier channels if reusable slots are not selected carefully or fail to allocate sufficient high-bit storage for clustered outliers. Therefore, the proposed stride-based adjacent outlier handling mechanism is necessary for maintaining accuracy while preserving the in-place property of InQuant.

D Algorithms and Layout Decoding

Algorithm 1 InQuant quantization for one KV-cache group

Require: KV group X, sampling set $\mathcal{S}$, saliency ratio ρ, group size G
Ensure: Packed 4-bit array P and group descriptor D_g

Estimate channel saliency S_c from sampled tokens $\mathcal{S}$ using the L1 score defined above.
Mark the top ρG channels as outlier channels $\mathcal{O}$; mark the remaining channels as reusable candidates $\mathcal{R}$.
Initialize descriptor D_g with the outlier bitmap, empty reuse map, and bit-width map.
for each channel c in channel order **do**
  if $c \notin \mathcal{O}$ **then**
    Quantize c to one 4-bit slot per value and pack into P.
  else
    Quantize c to 8-bit codes and split each code into high/low 4-bit nibbles.
    Store the high nibble at the original slot of c.
    if c has an unused low-saliency neighbor in $\mathcal{R}$ **then**
      Assign that neighbor as the reusable slot for c.
    else
      Identify the adjacent outlier group containing c and search unused candidates at stride distances from the group boundary.
    end if
    if a reusable slot is found **then**
      Store the low nibble in the reusable slot; update the reuse map and boundary hint in D_g.
    else
      Use the nearest descriptor-recorded aligned fallback slot and record the cross-group remap in D_g.
    end if
  end if
end for
return P, D_g

Algorithm 2 Descriptor-guided dequantization and recovery

Require: Packed array P, group descriptor D_g, scales/zero-points, sampled padding statistics
Ensure: Reconstructed KV group $\hat{X}$

Parse the outlier bitmap, bit-width map, reuse map, and optional remap entries from D_g.
for each channel c in channel order **do**
  if c is a normal channel **then**
    Decode its 4-bit slot and dequantize with the group scale/zero-point.
  else
    Read the high nibble from the original slot and the low nibble from the descriptor-selected reusable slot.
    Combine the two nibbles into an 8-bit code and dequantize the outlier value.
  end if
end for
for each reused low-saliency slot recorded in D_g **do**
  Reconstruct the overwritten low-saliency value using sampled padding statistics.
end for
Treat marker patterns only as boundary hints when they occur at descriptor-valid positions; otherwise decode them as normal 4-bit data.
return $\hat{X}$

Correctness of layout decoding. The descriptor, rather than the marker value itself, is the source of decoding correctness. For each group, the descriptor defines a one-to-one mapping from outlier channels to reusable slots or fallback remap entries. Dequantization therefore never searches the bitstream for arbitrary marker patterns; it reads only descriptor-valid boundary positions and treats all other 4-bit patterns as ordinary quantized data. This separation makes marker collision a performance issue rather than a correctness issue: even if a normal 4-bit value equals the marker pattern, it cannot change the decoded layout unless it appears at a descriptor-valid boundary and matches the recorded reuse map. Appendix E quantifies the remaining collision and recovery overhead.

E Bit-level Layout and Recovery Analysis

Detailed bit-level layout. InQuant uses a fixed 4-bit physical layout for compressed KV storage. For a normal channel, each quantized KV value occupies one 4-bit slot, and two consecutive 4-bit values are packed into one byte. For an outlier channel assigned to 8-bit precision, the 8-bit quantized code is split into two 4-bit nibbles. The first nibble is stored at the original outlier-channel slot, while the second nibble is placed into a reusable slot selected from a low-saliency channel. Therefore, InQuant does not introduce a separate outlier buffer and does not require channel permutation.

Information approximation in reused low-saliency slots. The storage borrowed from a reused low-saliency slot is not losslessly preserved. Instead, the corresponding 4-bit entry is overwritten by the extra outlier nibble, while the remaining entries in the reused channel are kept in their original 4-bit quantized form. During dequantization, the missing low-saliency entries are reconstructed using the sampling-based padding strategy. This design is based on the observation that low-saliency neighboring channels contribute much less to model accuracy than outlier channels. Therefore, InQuant trades a small amount of low-saliency information for a more faithful representation of high-saliency outliers.

Metadata and marker disambiguation. To decode the mixed layout, InQuant maintains a compact group-level descriptor for each quantization group. The descriptor records the outlier-channel bitmap, the reused-channel offset or stride, and the bit-width type. It is shared by all tokens in the same group and is amortized over the sequence length. Scale and zero-point are stored per quantization group, following standard group-wise KV quantization. In our implementation, the total metadata accounts for less than 0.1% of the compressed KV size.

Let G be the number of channels in a quantization group, L_g the number of tokens covered by the group descriptor, and m the number of selected outlier channels. The compressed data payload is $4GL_g$ bits. The descriptor contains an outlier bitmap of G bits, a bit-width map or threshold code, and m reuse entries. Each reuse entry stores either a local neighbor offset, a stride distance for an adjacent outlier group, or a fallback remap entry when the reusable slot crosses an aligned group boundary. Because the descriptor is shared by all tokens in the group, its amortized overhead is

$$\frac{G + O(m \log G)}{4GL_g},$$

which decreases with context length. This accounting is the basis for the physical compression ratio reported in the main tables: the 4-bit packed payload dominates the footprint, while descriptor and scale/zero-point metadata are included in the amortized memory measurement.

Markers are interpreted only at predefined channel-boundary metadata positions specified by the descriptor. A binary pattern is treated as a marker only if it appears at a valid marker position and is consistent with the group-level layout descriptor. The same pattern appearing in a normal 4-bit data slot is decoded as a regular quantized value. This positional constraint avoids ambiguity between markers and normal quantized values.

Recovery error and overhead. Table 7 reports the overhead and error of the recovery path. The average marker collision rate is only 0.03%, and the average padding reconstruction error is 0.31%. The cross-block remap overhead remains below 0.3% of end-to-end inference time. These results indicate that the approximation introduced by neighbor-slot reuse is small and does not introduce noticeable runtime overhead.

Marker collision rate measures the fraction of marker candidates that conflict with valid quantized values at marker-valid positions. Padding reconstruction error is computed as the relative L1 error between reconstructed and original low-saliency values. Remap overhead is measured as the additional runtime introduced by cross-block alignment divided by end-to-end inference latency.

F Fairness Analysis of Neighbor-slot Reuse

Motivation. InQuant differs from conventional mixed-precision KV-cache quantization because it does not allocate a separate high-precision outlier buffer. Instead, it stores the extra nibble of an 8-bit outlier value in a reused low-saliency slot. This design raises a natural fairness question: whether the

Table 7: Overhead and error analysis of marker-based recovery, padding reconstruction, and remap operations. Experiments are conducted on Mistral-7B with MMLU inputs under an 8K context length.

Metric	Avg.	Max.	Std.	Notes
Marker collision rate (%)	0.03	0.08	0.02	Marker decoding
Corrupted activation ratio (%)	0.06	0.11	0.03	Marker embedding
Padding reconstruction error (%)	0.31	0.48	0.07	Missing-value recovery
Remap overhead (%)	0.18	0.27	0.05	Cross-block remap
Dequantization overhead (ms)	1.42	2.16	0.31	Marker-based recovery

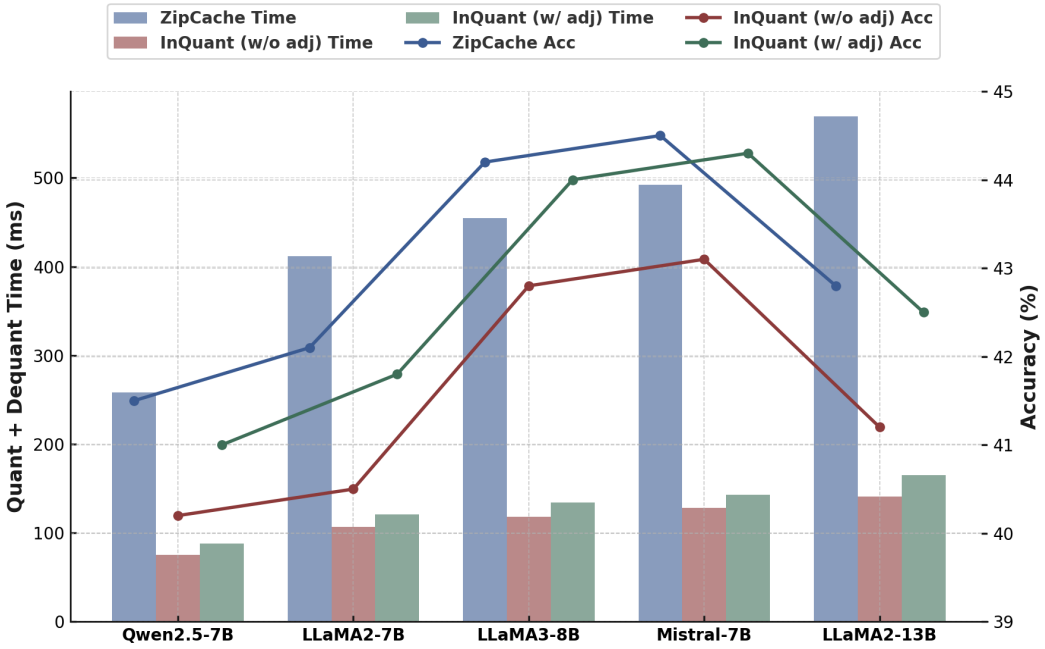

Figure 9: Across-model breakdown of adjacent outlier handling. We compare ZipCache, InQuant without adjacent-outlier handling, and full InQuant with stride-based adjacent-outlier handling across Qwen2.5-7B, LLaMA2-7B, LLaMA3-8B, Mistral-7B, and LLaMA2-13B. Bars report the total quantization and dequantization time, while lines report the corresponding accuracy. Full InQuant restores most of the accuracy loss caused by adjacent outliers while retaining a large latency advantage over ZipCache, showing that the proposed adjacent-outlier handling is effective across model architectures.

observed efficiency gain comes merely from overwriting less important KV-cache values. To answer this question, we construct several controlled baselines in Table 8.

Controlled baselines. Uniform INT4 evaluates the quality of pure 4-bit KV-cache quantization without special outlier preservation. INT4 + low-saliency padding keeps the same overwritten ratio as InQuant but does not use the overwritten slots to preserve outlier channels, isolating the effect of low-saliency reconstruction alone. Random neighbor reuse preserves outlier channels with the same 4-bit physical footprint but selects reusable slots randomly rather than according to channel saliency. InQuant w/o padding removes the reconstruction step for reused low-saliency slots, isolating the contribution of padding-based recovery. ZipCache is included as a strong mixed-precision baseline that preserves outliers without overwriting low-saliency values, but requires additional data movement and separate outlier handling.

Results. Uniform INT4 shows the largest quality degradation, confirming that outlier channels cannot be aggressively quantized together with normal channels. Adding low-saliency padding alone provides only marginal improvement, indicating that the main benefit of InQuant does not

Table 8: Detailed fairness analysis of neighbor-slot reuse on Mistral-7B. MMLU Avg. is averaged over STEM, Humanities, Social Sciences, and Other; GSM8K Avg. is averaged over Acc Norm. and Avg Acc.; LongBench Avg. is averaged over Qasper, HotpotQA, MultiFieldQA, GovReport, and NarrativeQA. All methods are evaluated with the same context length, serving configuration, and 20% saliency ratio when applicable.

Method	Physical CR	Outlier 8-bit Preserved	Saliency-aware Slot Reuse	Padding Recovery	Overwritten Ratio	MMLU Avg.	GSM8K Avg.	LongBench Avg. Qasper/HotpotQA/MultiFieldQA/GovReport/NarrativeQA	Quant. (ms)	Dequant. (ms)
Uniform INT4	4.0×	No	No	No	0%	52.8	36.0	35.1	56.4	41.8
INT4 + Low-saliency Padding	4.0×	No	Yes	Yes	20%	53.6	36.4	35.4	64.8	50.3
Random Neighbor Reuse	4.0×	Yes	No	Yes	20%	56.1	38.1	36.8	70.5	54.6
InQuant w/o Padding	4.0×	Yes	Yes	No	20%	57.6	39.5	37.7	76.9	52.1
InQuant Full	4.0×	Yes	Yes	Yes	20%	58.4	40.3	38.3	78.4	58.4
ZipCache	2.47×	Yes	Yes	–	0%	58.9	40.6	38.5	242.8	129.2

come from reconstructing low-saliency values. Random neighbor reuse improves over uniform INT4 by preserving outlier information, but its lower MMLU and LongBench scores show that random overwriting corrupts useful KV-cache entries. InQuant w/o padding further improves quality through saliency-aware slot reuse, but still trails full InQuant, demonstrating that padding-based recovery is necessary. Full InQuant achieves quality close to ZipCache while retaining a fixed 4-bit physical footprint and substantially lower quantization/dequantization latency. These results confirm that InQuant's gains come from the combination of outlier preservation, saliency-aware neighbor-slot reuse, and lightweight recovery, rather than from simply discarding low-saliency KV-cache information.

G Additional Breakdown of Adjacent Outlier Handling

We provide an additional across-model breakdown to further evaluate the effectiveness of the proposed stride-based adjacent outlier handling mechanism. While the main text focuses on the occurrence of adjacent outlier groups and the controlled adjacency-density analysis, this appendix studies whether the benefit of adjacent-outlier handling generalizes across different model architectures.

Figure 9 compares ZipCache, InQuant without adjacent-outlier handling, and full InQuant with stride-based adjacent-outlier handling across Qwen2.5-7B, LLaMA2-7B, LLaMA3-8B, Mistral-7B, and LLaMA2-13B. The bar plots report the total quantization and dequantization time, and the line plots report the corresponding accuracy. Compared with ZipCache, both InQuant variants substantially reduce quantization/dequantization time because they avoid explicit outlier separation and channel permutation. However, InQuant without adjacent-outlier handling suffers from accuracy degradation when adjacent outlier channels appear, since it cannot safely reuse immediate neighboring storage in clustered-outlier cases.

By incorporating stride-based adjacent-outlier handling, full InQuant recovers most of this accuracy loss. Although the stride-based search and marker-based recovery introduce a small amount of additional latency compared with the variant without adjacent handling, the total latency remains much lower than ZipCache across all evaluated models. These results confirm that adjacent outlier handling is not only useful in the controlled-density setting, but also provides consistent robustness across model architectures.

H LongBench Quality Evaluation

To verify that the efficiency gains of InQuant do not come at the cost of long-context quality, we additionally evaluate representative LongBench tasks using their original input lengths. We use LLaMA3-8B and Mistral-7B with a maximum context window of 32K, batch size 1, and 128 generated tokens. Inputs are not artificially padded to 32K; sequences longer than the context window, if any, are truncated to fit the 32K limit. The saliency ratio is set to 20%, where salient channels are quantized to 8 bits and regular channels are quantized to 4 bits, consistent with the main experiments. We report the task-specific LongBench score, where higher is better.

As shown in Table 9, InQuant maintains quality close to FP16 and the stronger mixed-precision baselines across question answering, multi-field comprehension, summarization, and narrative reasoning tasks. The largest gap between InQuant and ZipCache is within 0.3 points on most tasks, while InQuant avoids explicit outlier separation and channel permutation. These results indicate that the long-context latency and memory improvements reported in Figure 5 do not rely on sacrificing long-context recall quality.

Table 9: Long-context quality on representative LongBench tasks evaluated at their original input lengths with a 32K maximum context window. InQuant preserves quality close to FP16 and stronger mixed-precision baselines while providing lower latency and memory usage in the main long-context evaluation.

Model	Method	Qasper	HotpotQA	MultiFieldQA	GovReport	NarrativeQA
LLaMA3-8B	FP16	43.8	52.6	48.9	31.7	27.4
LLaMA3-8B	GEAR	39.6	47.8	44.1	29.3	24.8
LLaMA3-8B	MiKV	42.4	51.2	47.3	30.9	26.5
LLaMA3-8B	ZipCache	42.8	51.6	47.8	31.1	26.8
LLaMA3-8B	COMET	42.3	51.0	47.5	30.8	26.4
LLaMA3-8B	InQuant	42.5	51.3	47.6	30.9	26.6
Mistral-7B	FP16	41.6	50.8	46.7	30.9	26.1
Mistral-7B	GEAR	37.9	46.2	42.8	28.4	23.6
Mistral-7B	MiKV	40.4	49.5	45.6	30.1	25.3
Mistral-7B	ZipCache	40.8	49.8	45.9	30.3	25.5
Mistral-7B	COMET	40.5	49.4	45.7	30.0	25.2
Mistral-7B	InQuant	40.6	49.6	45.8	30.1	25.4

Table 10: Retrieval-style long-context recall evaluation on LLaMA3-8B. Needle denotes Needle-in-a-Haystack exact-match accuracy, and Passkey denotes RULER passkey retrieval accuracy. Higher is better.

Method	Needle 16K	Needle 32K	Passkey 16K	Passkey 32K	Avg.
FP16	99.5	98.8	99.2	98.4	99.0
GEAR	94.0	90.5	93.2	89.8	91.9
MiKV	98.0	96.6	97.4	95.9	97.0
ZipCache	98.5	97.4	98.0	96.8	97.7
COMET	98.0	96.9	97.6	96.2	97.2
InQuant	98.3	97.1	97.8	96.5	97.4

H.1 Retrieval-style Recall Evaluation

To further evaluate whether neighbor-slot reuse affects long-range recall, we conduct retrieval-style evaluations using Needle-in-a-Haystack and RULER passkey retrieval. Unlike general long-context benchmarks, these tasks directly test whether a model can recover a specific piece of information placed inside a long context. This is particularly relevant to InQuant because the method approximates low-saliency channels while preserving high-saliency outlier channels.

We evaluate LLaMA3-8B with context lengths of 16K and 32K. For Needle-in-a-Haystack, a target sentence is inserted at a random position in the context, and the model is required to recover the target answer. For RULER passkey retrieval, a key-value pair is placed in the input context, and the model is asked to output the corresponding passkey. We report exact-match accuracy over multiple random insertion positions. The saliency ratio is set to 20%, consistent with the main experiments.

As shown in Table 10, InQuant maintains retrieval accuracy close to ZipCache and other mixed-precision baselines. The small gap from FP16 indicates that the low-saliency approximation introduced by neighbor-slot reuse does not substantially damage long-range recall. In contrast, GEAR suffers a larger degradation because uniform low-bit quantization does not explicitly preserve outlier channels. These results complement the LongBench evaluation and further confirm that InQuant's latency and memory gains do not come at the cost of retrieval quality.

I Sensitivity Analysis

We evaluate Mistral-7B on MMLU to measure both accuracy and quantization/dequantization latency under varying saliency ratios, as summarized in Table 11. The results show that while increasing the saliency ratio generally leads to higher latency, the accuracy does not improve consistently.

Table 11: Performance comparison under different saliency ratios.

Saliency Ratio	Acc (%)	Q (ms)	DQ (ms)
10%	57.1	63	52
20%	**58.4**	**78**	**58**
30%	58.3	85	74
40%	57.1	108	89
60%	55.2	125	106

Table 12: Baseline scope for isolating InQuant's contribution. "Controlled" means that the method directly tests the in-place outlier-preserving layout under a comparable saliency ratio or physical footprint.

Method	KV-only	Mixed Precision	Tests Data Movement	Controlled	Main Role
ZipCache	Yes	Yes	Yes	Yes	Strong outlier-aware baseline
COMET	Yes	Yes	Yes	Yes	Channel-permutation baseline
MiKV	Yes	Yes	Partly	Yes	Token-saliency mixed-precision baseline
GEAR	Yes	No	No	Partly	Uniform low-bit reference
Uniform INT4 / variants	Yes	No / controlled	No	Yes	Physical-footprint fairness tests
KIVI	Yes	No	No	No	Complementary low-bit KV format
KVQuant	Yes	Partly	No	No	Complementary quantization datatype
QServe	No	Yes	System-level	No	Full serving-stack reference

Specifically, the 20% saliency ratio achieves the best overall trade-off, yielding the highest accuracy (58.4%) with moderate quant and dequant latencies (78 ms and 67 ms, respectively). Therefore, we adopt the 20% configuration in our main experiments.

J Additional Related Work and Baseline Discussion

Controlled-comparison principle. KV-cache quantization methods differ along several axes: quantization datatype, key/value granularity, outlier preservation, serving stack, and physical memory layout. A direct leaderboard-style comparison can therefore mix algorithmic accuracy gains with unrelated system choices. InQuant specifically targets the layout and data-movement overhead of outlier-aware mixed precision. For this reason, our main experiments compare against methods that expose this cost directly, and Table 2 compares controlled 4-bit-footprint variants that differ only in how overwritten low-saliency storage is selected and recovered. This isolates the contribution of neighbor-slot reuse from the choice of low-bit datatype or serving backend.

KIVI. KIVI proposes a tuning-free asymmetric 2-bit KV-cache quantization method. Its key observation is that the key cache and value cache exhibit different distributional characteristics: keys are better quantized per channel, while values are better quantized per token. Based on this observation, KIVI designs a hardware-friendly low-bit KV quantization scheme that substantially reduces KV memory usage. In contrast, InQuant focuses on a different problem: how to preserve high-precision outlier channels under an in-place physical layout. Instead of uniformly pushing the KV cache to very low precision, InQuant keeps salient channels at higher precision and reuses low-saliency slots to avoid separate outlier buffers or channel permutation.

KVQuant. KVQuant studies low-precision KV-cache quantization for very long-context inference. It introduces several techniques, including per-channel key quantization, pre-RoPE key quantization, non-uniform KV-cache datatypes, and dense-and-sparse outlier handling. These techniques are effective for improving low-bit quantization accuracy, especially in sub-4-bit settings. However, KVQuant still relies on specialized quantization formats and explicit handling of outlier components. InQuant is orthogonal to these quantization-format choices: its main contribution is an in-place mixed-precision storage layout that represents salient outliers without explicit outlier separation or channel permutation. Therefore, KVQuant-style quantization formats could potentially be combined with InQuant's in-place layout in future work.

QServe. QServe is a serving-oriented quantization system based on W4A8KV4 quantization and system co-design. It optimizes both quantization algorithms and GPU kernels, including low-overhead dequantization, SmoothAttention, and serving-backend optimizations. QServe demonstrates that KV-cache quantization must be co-designed with the inference system to translate memory savings into real throughput gains. InQuant shares this system-level motivation, but targets a complementary

Table 13: Comparison with closely related KV-cache quantization and LLM serving systems. The table summarizes the main design granularity, outlier handling strategy, layout requirement, and the role of each method in our evaluation or discussion.

Method	Granularity	Handles Outliers	Needs Permutation or Extra Buffer	KV-only	Role in This Paper
KIVI	Per-channel key / per-token value asymmetric	Yes	No / limited metadata	Yes	Appendix discussion
KVQuant	Non-uniform / data-free KV quantization	Yes	No explicit permutation	Yes	Appendix discussion
QServe	W4A8KV4 serving system co-design	Yes	System-level layout and kernel co-design	No	Appendix discussion
ZipCache	Salient token / channel-aware KV quantization	Yes	Separate outlier handling	Yes	Main baseline
COMET	Block-wise / channel permutation	Yes	Yes	Yes	Main baseline
InQuant	In-place neighbor slot reuse	Yes	No permutation; no separate outlier buffer	Yes	Proposed method

bottleneck. Rather than designing a full W4A8KV4 serving stack, InQuant specifically reduces the data movement and layout-transformation overhead introduced by outlier-aware mixed-precision KV-cache quantization.

Why they are not included in the main comparison. The main comparison focuses on methods that are closest to InQuant's setting: outlier-aware or mixed-precision KV-cache quantization methods that assign different precision to salient and regular KV channels. KIVI and KVQuant mainly study low-bit KV quantization formats and key/value-specific quantization granularities, while QServe is a full serving system that jointly quantizes weights, activations, and KV cache. These systems are highly relevant but are not strictly controlled baselines for isolating the effect of InQuant's in-place mixed-precision layout. We therefore discuss them here and leave a full integration with these low-bit quantization formats and serving backends as future work.

Discussion. Table 13 summarizes several closely related KV-cache quantization and LLM serving systems. KIVI and KVQuant are highly relevant KV-cache quantization methods that reduce KV memory through low-bit or non-uniform quantization. Their primary focus is improving quantization accuracy under aggressive KV compression, whereas InQuant focuses on reducing the runtime overhead of mixed-precision KV-cache compression by avoiding explicit outlier separation, channel permutation, and additional outlier buffers.

Compared with ZipCache, which preserves salient KV entries through separate handling, and COMET, which improves mixed-precision locality through channel permutation, InQuant preserves high-saliency outlier channels within a fixed 4-bit physical layout by reusing low-saliency neighboring slots. This design eliminates explicit channel permutation and avoids a separate high-precision outlier buffer, which explains the lower quantization and dequantization latency observed in the main experiments.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the scope of InQuant as an in-place mixed-precision KV-cache quantization method, including its physical 4-bit footprint, neighbor-slot reuse mechanism, and empirical accuracy/latency claims. The claims are supported by the main experiments and the limitations section clarifies the evaluated model sizes, context lengths, and hardware scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a Limitations section that discusses the assumptions behind low-saliency slot approximation, the need for descriptor-aware kernels, and the current evaluation scope. It also identifies larger models, additional GPU architectures, longer contexts, and integration with sub-4-bit formats as future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results requiring theorem statements or proofs. It provides algorithm descriptions, layout invariants, and amortized metadata analysis in the method and appendix instead.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper discloses the models, datasets, quantization settings, saliency ratio, sampling ratio, latency measurement protocol, hardware setup, and baseline configuration. Additional reproducibility details and algorithms are provided in the appendix, and the supplementary material includes anonymized source code and scripts for reproducing the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide anonymized source code in the supplementary material, including CUDA kernels, evaluation scripts, configuration files, and instructions for reproducing the main accuracy, latency, memory, and serving-breakdown results. Upon acceptance, we will release the full codebase publicly with documentation.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the evaluated models, benchmarks, context lengths, generated-token lengths, batch sizes, saliency ratio, sampling ratio, quantization configuration, warm-up runs, measured runs, and hardware. Additional implementation and reproducibility details are included in the appendix and supplementary artifact.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Latency and memory results are averaged over five measured runs after two warm-up runs, and deterministic decoding is used for accuracy evaluation to reduce run-to-run variability. Where applicable, the paper reports variability information such as standard deviations for adjacent-outlier statistics.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the main hardware platform, including an Intel Xeon 96-core CPU and a single NVIDIA A100 80GB GPU, together with batch size, context length, generation length, and measurement protocol. The supplementary artifact provides commands and environment information for reproducing the reported experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. It uses publicly available models and benchmarks, does not involve human subjects, and does not introduce new high-risk datasets or models.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper includes a Broader Impact discussion covering positive impacts such as reducing hardware and energy costs for long-context LLM serving, as well as potential negative impacts such as enabling cheaper large-scale misuse of existing LLM capabilities. It also recommends safety, privacy, and monitoring mechanisms for deployment.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new pretrained language model, image generator, scraped dataset, or other high-risk asset. The released artifact is an anonymized implementation of an inference-efficiency method and does not by itself introduce new generative capabilities.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the existing models, datasets, and baseline methods used in the evaluation, including public LLMs and standard benchmarks. The supplementary artifact documents the corresponding software dependencies, versions, and applicable licenses or terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces a new code asset for InQuant, and the supplementary material provides an anonymized implementation with documentation and reproduction scripts. The full codebase will be officially open-sourced with license information upon acceptance.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method does not use LLMs as an important, original, or non-standard component of the research process. LLMs are the target models evaluated for KV-cache quantization, and any routine writing or formatting assistance, if used, does not affect the methodology, scientific rigor, or originality of the work.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.