KITTY: ACCURATE AND EFFICIENT 2-BIT KV CACHE QUANTIZATION WITH DYNAMIC CHANNEL-WISE PRECISION BOOST

Haojun Xia [*1] **Xiaoxia Wu** [*2] **Jisen Li** [*3] **Tsai-chuan Wu** [2] **Junxiong Wang** [2] **Jue Wang** [2] **Chenxi Li** [2] **Aman Singhal** [2] **Alay Dilipbhai Shah** [2] **Alpay Ariyak** [2] **Donglin Zhuang** [1] **Zhongzhu Zhou** [1] **Ben Athiwaratkun** [2] **Zhen Zheng** [4] **Shuaiwen Leon Song** [2]

ABSTRACT

The KV cache is a dominant memory bottleneck for LLM inference. While 4-bit KV quantization preserves accuracy, 2-bit often degrades it, especially on long-context reasoning. We close this gap via an algorithm–system co-design for mixed-precision KV caching: *Kitty*. On the algorithm side, extensive experiments show that *Dynamic Channel-wise Precision Boost* — which ranks Key-cache channels by sensitivity and keeps only a small fraction at higher precision — maintains near-zero drop in accuracy while approaching 2-bit memory. On the system side, the primary challenge lies in managing these dynamic 4-bit channel boosts without compromising memory efficiency or the execution speed of attention layers. *Kitty* addresses this through a hardware-aware memory layout and highly optimized system designs, ensuring that our on-the-fly KV quantization incurs negligible runtime overhead while maximizing memory footprint reduction. This synergistic design allows Kitty to unlock the full potential of 2-bit quantization without sacrificing real-time inference throughput. Specifically, *Kitty* addresses these issues by decomposing each mixed-precision Key page into two tensors with unified 2-bit precision. Based on this, Kitty provides a page-centric KV layout, Triton-compatible page dequantization kernels, and a lightweight runtime pipeline that reduces and amortizes the runtime overhead. Across seven tasks and two model families (Qwen3, LLaMA3), *Kitty* cuts KV memory by nearly $8\times$ with negligible accuracy loss, enabling up to $8\times$ larger batches and $2.1\times$–$4.1\times$ higher throughput under the same memory budget. We release the full implementation of Kitty at https://github.com/Summer-Summer/Kitty.

1 INTRODUCTION

As large language models (LLMs) advance, their ability to process extremely long contexts (e.g., 128K tokens (OpenAI et al., 2024; Dubey et al., 2024)) has enabled powerful applications such as detailed document understanding, extended dialogues, and complex reasoning (e.g., chain-of-thought (Wei et al., 2024)). However, their progression has exposed a severe systems bottleneck: the enormous memory footprint of the KV cache. Unlike model weights (which are static), the size of the KV cache grows proportionally with both context length and batch size, leading to outsized GPU memory footprints in long-context inference. For a model like LLaMA3-70B (Dubey et al., 2024), serving 32 requests with a 128K sequence requires more than 1.2 TB of KV cache storage, which is nearly an order of magnitude larger than the model weights themselves. But state-of-the-art data-center GPUs such as the NVIDIA B200 GPUs provide only 192 GB of memory and cost over $30,000 each, making such deployments prohibitively expensive. The massive KV cache further exacerbates inference latency due to the heavy data movement between GPU memory and compute units.

KV cache quantization offers a compelling direction to address this gap, especially the post-training scheme which can be applied without re-training or fine-tuning. Moreover, with proper system support (e.g., Du et al. (2025)), it can effectively reduce both memory consumption and inference latency. Unlike token pruning, quantization preserves all contextual information without discarding tokens. Nevertheless, low-bit (e.g. 2-bit) KV cache quantization remains challenging in practice. Our empirical investigation shows that quantizing the KV cache to 4-bit precision using the state-of-the-art method KIVI (Liu et al., 2024) can maintain accuracy. However, further reducing to 2 bits significantly harms model accuracy across a range of downstream tasks. These findings suggest that more nuanced approaches are required to unlock the full potential of KV compression at even lower precision.

[*]Core contributors [1]University of Sydney [2]Together AI [3]University of Illinois Urbana-Champaign [4]Microsoft. Correspondence to: Shuaiwen Leon Song <leon@together.ai>, Zhen Zheng <zhengzhen@microsoft.com>.

Proceedings of the 9^{th} MLSys Conference, Bellevue, WA, USA, 2026.

To address this problem, we propose a novel approach for 2-bit KV cache quantization, which we call *channel-wise precision boost*. The key insight is that applying a uniform precision across the entire KV cache is suboptimal. Orthogonal to prior work (Zhang et al., 2024b), which preserves **important tokens** in higher precision, *channel-wise precision boost* preserves only **critical channels** in higher precision while aggressively compressing the remainder. Our method is motivated by the observation that certain *critical channels* within the key cache play a disproportionately important role in maintaining model accuracy. Based on this novel method, we propose a systematic quantization algorithm, *Kitty*, shown in Figure 1. Furthermore, we design and build an end-to-end inference system for our novel 2-bit KV quantization algorithm on GPU platforms. To support our novel KV cache memory layout, we developed GPU kernels with Triton (Tillet et al., 2019) for: (1) efficient quantization of newly generated KV vectors and cache updates, and (2) efficient execution of attention mechanism. Crucially, Kitty embodies a algorithm-system co-design: the quantization algorithm ensures superior model fidelity and memory efficiency, while the underlying system architecture effectively masks the dynamic overhead to unlock high-throughput inference.

This paper makes the following contributions:

- We observe that state-of-the-art 2-bit KV cache quantization methods like KIVI (Liu et al., 2024) substantially degrade the accuracy of reasoning LLMs, revealing a critical representation bottleneck for low-bit KV cache quantization.
- We introduce *Dynamic Channel-wise Precision Boost*, a novel 2-bit quantization algorithm for KV cache, inspired by the key observations in *channel-wise patterns* and *channel-wise quantization sensitivity*. Combined with other optimizations, we propose a systematic KV cache quantization scheme called **Kitty**. Extensive experiments show that Kitty can significantly outperform prior works in accuracy.
- We propose novel system designs and build an end-to-end inference system for our novel 2-bit KV quantization algorithm on GPU platforms. This system desgin solve the channels of handling dynamic 4-bit channel boosts while keeping the page layout coalesced and the dequantization uniform. Our inference system enables $8\times$ larger batch sizes and achieves $2.1\times \rightarrow 4.1\times$ higher inference throughput compared to the FP16 baseline while maintaining the same memory budget.

The remainder of this chapter is organized as follows. Section 2 details the background and motivation. Section 3 describes our algorithm design for 2-bit KV cache quantization. We present our system designs in Section 4. Section 5 details our experimental results of inference accuracy and throughput.

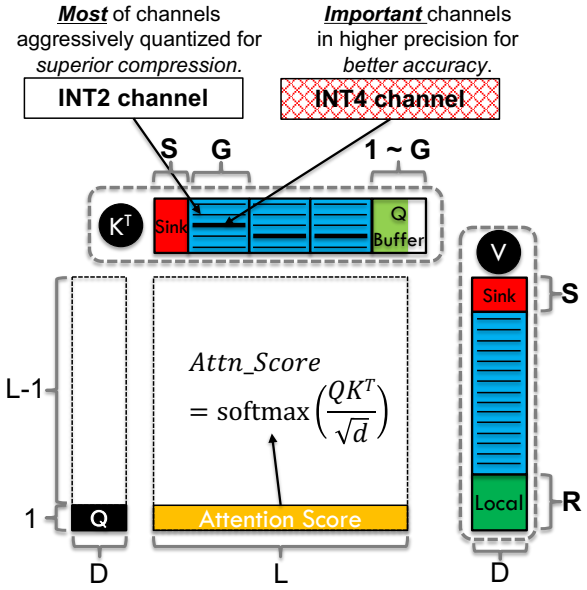


Figure 1. Illustration of our KV cache quantization scheme, **Kitty**. The figure shows the decoding phase, where only the current *query* vector (a single token) participates in computation, while all previously stored *key* and *value* vectors in the KV cache are reused. The key cache is organized into three parts: (i) *Sink* (initial tokens kept in FP16), (ii) a *Q-Buffer* (quantization buffer, temporarily storing the FP16 KVs before forming a quantization group), and (iii) *quantized channels* (Most channels quantized to INT2 for maximum compression, a small fraction of channels preserved in INT4 for accuracy preservation). The value cache is quantized per-token with a sliding window, where both *Sink* (initial tokens) and the most recent *Local* (local tokens) are retained in FP16. Here, L denotes the sequence length and D the head size; S is the number of preserved sink tokens, G the quantization group size, and R the local window size. The default configuration is: $S = 32$, $R = 128$, $G = 128$, which provides a good balance between accuracy and memory savings.

2 Background & Motivation

The KV cache can help avoid the expensive recomputation of historical key and value tensors by retrieving them from cache memory, significantly accelerating inference. However, the cache size grows linearly with context length. As real-world applications scale, the KV cache quickly dominates memory usage and becomes a critical bottleneck for inference.

2.1 Existing KV Quantization Strategies

Low-bit quantization has emerged as a promising direction to reduce the memory footprint of the KV cache. The con-

cept is to convert full-precision keys and values (e.g., FP16) into compact representations such as 8-bit, 4-bit, or even 2-bit formats. Unlike weight quantization, which is static and applied once after training, KV cache quantization must be performed dynamically during runtime, making the problem more challenging.

Per-token vs. Per-channel Quantization. The Key or Value cache can be represented as a matrix in shape (B, H, L, D), where B is the inference batch size, H is the number of KV head, L is the sequence length, and D is hidden size for each head. Thus, for each head of a certain request, the KV cache has shape (L, D), which is a 2-D dimensional matrix consisting of L tokens and each token is a vector of size D.

As a result, the KV cache can be quantized along two different dimensions, L and/or D. Per-token quantization applies a separate scale (and optionally zero-point) to each token's key or value vector, so each token can be quantized separately. Per-channel quantization instead computes a scale per channel, shared across all tokens, and has been observed to preserve accuracy better for Key cache quantization (Liu et al., 2024; Hooper et al., 2024; Zhang et al., 2024a). In practice, most recent works adopt per-channel quantization for the K cache and per-token quantization for the V cache.

Mixed-precision Quantization. Quantizing the entire KV cache to a very low precision (e.g., 2-bit) typically hurts model quality severely. To mitigate this, hybrid schemes preserve a subset of elements in higher precision while quantizing the rest. For example, KVQuant (Hooper et al., 2024) identifies outliers and stores them in FP16 with a sparse representation. However, KVQuant is not hardware-friendly and usually suffers from low system-level efficiency, since it introduces additional runtime overhead from sparse-dense multiplications, which could be slow on GPUs (Xia et al., 2023; Gale et al., 2020). KIVI (Liu et al., 2024) and BitDecoding (Du et al., 2025) retain the most recent tokens in full precision to preserve an accurate local context, but they still suffer from severe accuracy degradations in low-precision regimes (e.g. 2-bit). MiniKV (Sharma et al., 2024) proposes a layer-discriminative bit allocation scheme for each layers. KVTuner (Li et al., 2025) proposes to tune layer-wise mixed precision bitwidths for the KV cache. QuaRot (Ashkboos et al., 2024) applies orthogonal rotations to activations and KV cache so that the distributions become outlier-free. Several recent studies have explored mixed-precision quantization for weights and activations (Zheng et al., 2024; Zhao et al., 2024); however, these approaches are orthogonal to the focus of this work.

KIVI (Liu et al., 2024) is a tuning-free, 2-bit KV framework, serving as an important algorithm baseline in this paper. It relies on the insight that Keys and Values exhibit different outlier patterns. Accordingly, it applies *per-channel* quantization for Keys and *per-token* quantization for Values. Meanwhile, it also maintains a small FP16 sliding window for both Key and Value cache to preserve local context. Comparing to Figure 1, KIVI does not preserve FP16 Sink and has no INT4 channels.

Non-KV Quantization. While our focus is on KV quantization, it is worth noting that several techniques for weight and activation quantization are orthogonal and potentially complementary (Xiao et al., 2024a; Lin et al., 2024).

2.2 Motivation: Accuracy Drop for Low-bit KV

Despite progress, existing solutions struggle to maintain accuracy when pushing the precision of KV cache below 4 bits. Liu et al. (2025) shows that quantizing KV cache to 3 bits already causes significant accuracy degradations with state-of-the-art quantization algorithms (Hooper et al., 2024; Ashkboos et al., 2024). It follows that 2-bit precision should be even more challenging.

We empirically observe that while 4-bit KV cache quantization can match FP16 closely, reducing it further to 2-bit (INT2) leads to substantial accuracy degradation across reasoning and generation benchmarks.

As Table 1 shows, reducing the KV cache precision from 16 to 4 bits causes almost no degradation in accuracy for Qwen3-8B and LLaMA3-8B. However, further reducing the cache to 2 bits greatly deteriorates the average drops to -15.23 and -10.15 respectively. These results highlight that simply applying existing quantization methods to KV cache is insufficient, and thus motivate a line of inquiry: **can more nuanced techniques mitigate these losses and enable practical low-bit KV cache quantization for long-context LLM inference?**

Table 1. Accuracy degradation of low-bit KV cache quantization on Qwen3-8B (Yang et al., 2025) and LLaMA3-8B (Dubey et al., 2024). While 4-bit KIVI (Liu et al., 2024) maintains accuracy, 2-bit KIVI shows a significant drop. *Note:* "MATH" and "GPQA" denote the *MATH-Algebra* and *GPQA-Diamond* subsets.

Model	Task	FP16	KIVI-4bit	KIVI-2bit
Qwen3-8B	GSM8K	94.79	94.41	89.13
	MATH	88.26	88.46	47.29
	GPQA	40.71	38.98	32.24
	HumanEval	84.82	84.09	76.89
	AIME24	71.67	77.67	57.00
	AIME25	66.00	65.33	52.33
LLaMA3-8B	GSM8K	76.75	76.09	63.58
	MATH	47.15	47.85	31.45
	GPQA	26.94	25.82	23.88
	HumanEval	63.96	62.07	55.30

3 ALGORITHM DESIGN

As discussed in Section 2.2, pushing KV cache quantization to extremely low precision (e.g. INT2) to save memory comes at the cost of a wide accuracy gap. In this section, we first present our *design space exploration* in Subsection 3.1, where we try to reduce this gap with straightforward optimizations. Then, we propose *channel-wise precision boost* in Subsection 3.2. We present the overall quantization scheme in Subsection 3.4.

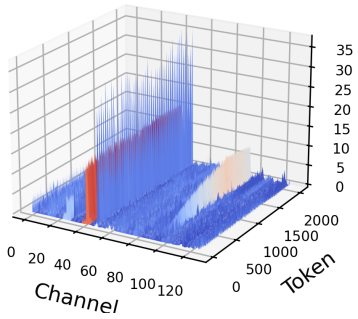
3.1 Design Space Exploration

Preserving the Initial Tokens in Full Precision The initial tokens in a sequence are sometimes referred to as *attention sinks*, as noted in StreamingLLM (Xiao et al., 2024b). These tokens typically receive disproportionately high attention weights and thus contribute significantly to the final outputs. In this paper, we stipulate that preserving the initial tokens in full precision can help mitigate accuracy loss with negligible overhead. We implemented KIVI-K2V2*, which is an algorithmic variant of KIVI, where the first 32 tokens are additionally preserved in full precision in KV cache. Empirical results in Table 2 show that KIVI-K2V2* substantially alleviates accuracy degradation across benchmarks. On average, KIVI-K2V2* outperforms the original KIVI-K2V2 by +8.28 and +5.33 on Qwen3-8B and LLaMA3-8B respectively.

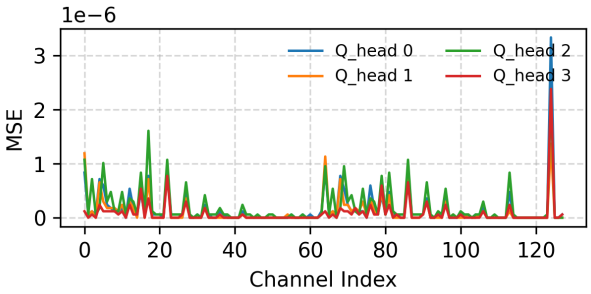
Increasing the Precision of Key Cache. While preserving the precision of sink-token entries recovers some performance, this technique still leaves much to be desired when compared to the FP16 baseline. To further reduce this gap and better understand the relative importance of the Key and Value caches, we also evaluate two design variants: KIVI-K2V**4*** and KIVI-K**4**V2*. Results in Table 2 indicate that increasing the precision of the Key cache proves substantially more effective than doing so for the Value cache. In particular, KIVI-K**4**V2* outperforms KIVI-K2V**4*** and achieves accuracy approaching the FP16 baseline across multiple benchmarks, underscoring the importance of increasing the precision of Key cache over the Value cache.

3.2 Channel-wise Precision Boost

In the previous subsection (§ 3.1), we observe that raising the precision of the Key cache can effectively mitigates the accuracy degradation from KV cache quantization. It motivates a key question: *rather than boosting the precision of the entire Key cache, can we obtain comparable accuracy gains by increasing the precision of only a small subset of the Key cache?* To explore this idea, we first perform both visual and statistical analysis of the Key-cache tensors, and then propose our key innovation according to the observations.



(a) Visualization of Key-cache.



(b) MSE error on attention score after quantizing each channel.

Figure 2. Visual and statistical analysis of Key cache from Layer 10, Qwen3-8B. (a) Visualization of Key-cache magnitude from the first KV head. The uneven distribution, with a few channels showing consistently high magnitudes, motivates a channel-aware approach to quantization. The vertical axis denotes activation magnitudes, while the horizontal axis spans the *token* and *channel* dimensions. (b) The mean squared error (MSE) between the original attention score matrix with its perturbed counterpart after quantizing each channel of Key cache. The pattern is consistent between different Q heads who share the same Key cache due to grouped-query attention (Ainslie et al., 2023).) Similar patterns are observed on other layers/models.

Observation-1: Channel-wise Patterns in Key Cache. We visualize the absolute values of Key-cache activations across multiple layers on Qwen3-8B, and find that the magnitudes of different channels in Key cache vary considerably. As shown in Figure 2a, a subset of channels consistently exhibits higher magnitudes. While this figure only presents the visualization of layer 10, similar pattern is observed on other layers. These observations align with prior work (Liu et al., 2024; Hooper et al., 2024), which leveraged per-channel quantization to reduce quantization errors.

These observations suggest that not all channels in the Key cache behave equally. Some channels appear to exert a stronger influence, while others may have only a marginal effect. This uneven contribution motivates a channel-aware perspective: by selectively allocating higher precision to

Table 2. Accuracy comparison across KV cache schemes on Qwen3-8B (Yang et al., 2025) and LLaMA3-8B (Dubey et al., 2024). Rows are tasks (transposed from original tables); columns are methods. Within each model block, the best per-row is in **bold**. *Note:* "MATH" = MATH-Algebra, "GPQA" = GPQA-Diamond. Algorithm names ending with an asterisk (*) indicate that the initial 32 tokens are preserved in full precision.

Model	Task	FP16	KIVI-K2V2	KIVI-K2V2*	KIVI-K2V4*	KIVI-K4V2*
Qwen3-8B	GSM8K	94.79	89.13	89.71	90.14	**93.96**
	MATH	88.26	47.29	74.92	82.50	**87.92**
	GPQA	40.71	32.24	36.02	35.82	**40.51**
	HumanEval	84.82	76.89	78.54	81.77	**83.41**
	AIME24	71.67	57.00	67.67	71.00	**76.00**
	AIME25	66.00	52.33	57.67	**64.33**	**64.33**
LLaMA3-8B	GSM8K	76.75	63.58	71.04	72.38	**76.62**
	MATH	47.15	31.45	44.12	44.09	**48.41**
	GPQA	26.94	23.88	23.67	24.80	**25.20**
	HumanEval	63.96	55.30	56.71	59.02	**62.80**

the most influential channels, it may be possible to preserve model accuracy while still reducing the overall KV cache memory footprint.

Observation-2: Channel-wise Quantization Sensitivity. To understand the importance of each channel, we isolate the impact of quantization on each individual channels. Specifically, we quantize a single channel to 2 bits (INT2) each time while keeping other channels unchanged, and calculate its impact on the attention score:

$$\text{attn\_score} = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right). \tag{1}$$

We compute the **mean squared error (MSE)** between the original attention score matrix with its perturbed counterpart (after quantizing a channel), and use this MSE as the metric to quantify the sensitivity of each channel in Figure 2b. Since Grouped-Query Attention (GQA) (Ainslie et al., 2023) is used in these models, where one key head is shared across multiple query heads, we report the MSE for each query head separately with different colors.

As shown in Figure 2b, different channels exhibit vastly different levels of sensitivity to quantization. A small fraction of channels consistently introduce larger errors to the attention score upon 2-bit quantization, suggesting they should be preserved in higher precision, e.g. 4-bit. Meanwhile, quantizing other channels to 2-bit causes less significant error on the attention score, indicating that they can be quantized more aggressively to 2-bit safely. Similar patterns are observed on other layers and models. This disparity highlights a clear opportunity: rather than uniformly quantizing all channels to 2-bit, selectively preserving a small fraction of sensitive channels in higher precision is promising to preserve inference accuracy while reducing the size of KV cache. This insight forms the foundation of the following key innovation.

Key Methods: Boosting Precision of Critical Channels Inspired by prior observations, we propose our *channel-wise precision boost*, where the critical channels are preserved in higher precision (INT4) while the others are aggressively quantized to lower precision (INT2), to reduce the distortion introduced by uniform 2-bit quantization and improving inference accuracy.

To apply our *channel-wise precision boost* method, we need to first identify a subset of channels to boost. Directly computing the sensitivity of every channel at runtime, as we did in Figures 2b, is prohibitively expensive which requires computing the attention score matrix and measuring its MSE against an FP16 baseline for each channel. To address this, we approximate channel importance using lightweight heuristics that can be computed in a single pass. In this work, we use *magnitude-based selection* heuristic to approximate the channel importance. We therefore define the importance score as the average magnitude of channel i across all tokens:

$$s_i = \frac{1}{T}\sum_{t=1}^{T} |x_{i,t}|. \tag{2}$$

Note that the score of each channel is required to be computed during inference runtime. Based on these channel-wise scores, top-K [1] channels are selected and stored to KV cache in higher precision (e.g., INT4). It is worth noting that our channel selection is **dynamic** and **input-adaptive**, rather than a static configuration. Specifically, we implement this at a granular level: for each quantization group (e.g., every 128 tokens), the saliency scores of each channel are recomputed on-the-fly. This allows our strategy to

[1] K is determined by the anticipated memory budget, e.g. if you want to boost 25% of the channels to higher precision and the size per head is 128, then K is 32.

identify and boost critical channels based on the real-time distribution of KV values, ensuring robustness across varying input patterns.

3.3 Formal Analysis of Error-bound

To provide a deeper algorithmic justification, we incorporate a formal theoretical derivation in this subsection. We first analyze the error propagation from the quantized Key cache to the attention logits P and derive the perturbation ΔP as a weighted sum of per-channel quantization noise E. Then, we provide a formal proof demonstrating that high-magnitude channels dominate the total error bound and describe how *channel-wise precision boost* helps reducing the quantization error. Finally, We further justify that while the query Q is dynamic, targeting the intrinsic R_d via magnitude-based selection minimizes the worst-case error, ensuring robust performance across diverse and unpredictable prompts.

Quantization Error Propagation. Let $P = QK^T$ denote the attention scores in FP16 precision. When the Key cache is quantized, the quantized Key is represented as $\hat{K} = K + E$, where $E \in \mathbb{R}^{T\times D}$ is the quantization noise tensor. The perturbation in the attention score ΔP is given by:

$$\Delta P = Q\hat{K}^T - QK^T = QE^T. \tag{3}$$

For a specific query vector Q_i and key vector K_j, the absolute error $\Delta P_{i,j}$ is bounded by the weighted sum of per-channel quantization errors $E_{j,d}$:

$$|\Delta P_{i,j}| = \left|\sum_{d=1}^{D} Q_{i,d}E_{j,d}\right| \le \sum_{d=1}^{D} |Q_{i,d}| \cdot |E_{j,d}|. \tag{4}$$

Error Reduction via Channel-wise Precision Boost. In uniform quantization, the error for channel d is bounded by half of its *quantization step size*[2] Δ_d, while the step size is determined by the dynamic range R_d and the bit-width b of that quantization group:

$$|E_{j,d}| \le \frac{\Delta_d}{2}, \quad \text{where} \quad \Delta_d = \frac{R_d}{2^b - 1}. \tag{5}$$

Specifically, R_d denotes the dynamic range (the span between maximum and minimum values) of channel d within a specific quantization group. As observed in our empirical analysis (Fig 2a), the dynamic range R_d exhibits significant *inter-channel disparity*. In a uniform 2-bit setting ($b = 2$), channels with large R_d result in excessively large Δ_d, which dominates the total error bound in Eq. (2). By selectively boosting these critical channels to 4-bit ($b = 4$), their individual step sizes are reduced by a factor of:

$$\frac{\Delta_{d,\text{2-bit}}}{\Delta_{d,\text{4-bit}}} = \frac{2^4 - 1}{2^2 - 1} = \frac{15}{3} = 5\times. \tag{6}$$

This targeted refinement yields a disproportionate reduction in the total error bound $|\Delta P_{i,j}|$ with minimal memory overhead compared to a global bit-width increase.

Robustness of Magnitude-based Selection. While the theoretical error bound is influenced by the dynamic query $Q_{i,d}$, a robust serving system must minimize the potential impact across diverse inputs where Q is unknown a priori. We prioritize the reduction of Δ_d as it represents the "worst-case" noise contribution of a channel. Furthermore, the observed *intra-channel similarity* ensures that the average magnitude (L_1-mean) is a stable and robust proxy for the dynamic range R_d. Unlike the Max-Min range (L_∞), which is sensitive to transient outliers, the average magnitude captures the persistent energy of a channel over the entire page. This allows Kitty to consistently protect the most impactful dimensions of the KV cache while maintaining $O(N)$ computational efficiency.

3.4 Kitty: Overall Quantization Scheme

Our **Kitty** quantization algorithm builds upon the foundation of KIVI (Liu et al., 2024) while introducing two key enhancements to further improve accuracy under aggressive low-bit settings. (1) We preserve the initial tokens (*Sink tokens*) in full precision for both the Key and Value caches. (2) More importantly, our novel *channel-wise precision boost* mechanism is applied to our Kitty. Figure 1 illustrates the overall design of our quantization scheme, Kitty, which integrates all the above optimizations into a unified scheme.

To evaluate the accuracy impact of this quantization algorithm, we integrated *Kitty* quantization scheme into the HuggingFace Transformers framework (Hugging Face Team, 2025), and built an accuracy simulation framework to validate the accuracy impact of Kitty.

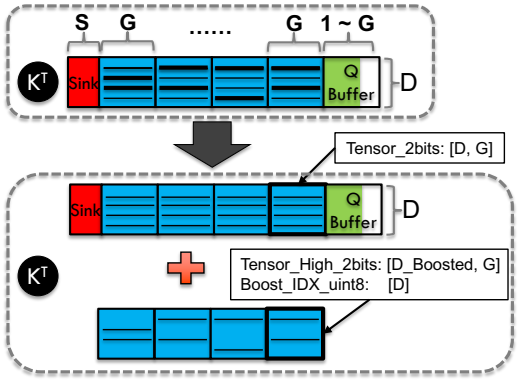
4 System Design and Implementation

This section describes the system-level innovations that enable Kitty to efficiently execute mixed-precision quantization on GPUs. Our design includes: (1) a page-centric memory layout for Kitty KV Cache, (2) Triton compatible GPU kernel design for fused dequantization and matrix multiplication, and (3) a lightweight runtime pipeline for Kitty attention execution. The system and algorithm are **inextricably coupled**: the algorithm creates the opportunity for massive throughput but introduces memory irregularity; the system (page-centric layout, amortized quantization overhead, fused dequantization) resolves this irregularity.

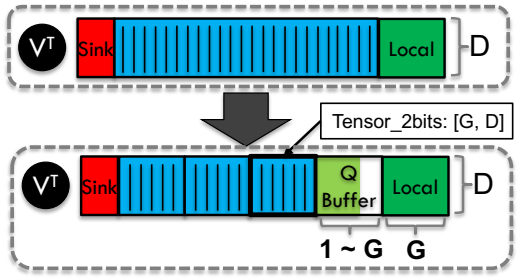
[2]The quantization step size Δ_d represents the numerical resolution of the quantized format; a smaller Δ_d indicates a finer-grained representation of the original floating-point values.

4.1 Page-Centric Memory Layout.

To support our proposed Kitty algorithm, we design a memory- and compute-efficient layout for the KV cache. Crucially, our page-centric layout maintains seamless compatibility with industry-standard *paged attention* mechanisms (Kwon et al., 2023), which is essential for scalable cache management and for reducing memory fragmentation. Specifically, we implement this feature by treating each quantization group (e.g., 128 tokens) as an independent page. Moreover, each Key cache page is decomposed into two 2-bit matrices: a dense matrix and a structured sparse matrix[3], which are stored separately. This structural consistency—both matrices now contain only 2-bit elements—simplifies the orchestration of the efficient runtime execution and memory management detailed in Sections 4.2 and 4.3.



(a) Memory layout of key cache.



(b) Memory layout of value cache.

Figure 3. Illustration of Kitty's page-centric KV cache layout.

Paged and Quantized Key-Value Cache. As shown in Figure 3, we store the KV cache with several full-precision buffers (i.e. *Sink*, *Q-Buffer*, *Local*) in GPU memory and many fixed-size memory pages (inspired by Paged Attention (Kwon et al., 2023)). In the design of Kitty shown in Figure 1, G (e.g. 128) tokens form a quantization group in the Key cache, we naturally choose G as the page size. Thus, each page contains (D, G) elements, where D is `size_per_head`. For efficient memory management, we also split the Value cache into pages.

[3]We do not store the zero elements within the sparse matrix.

Dynamic Dense-Sparse Decomposition of Quantized Pages. During attention computation, all pages of the KV cache must be fetched from HBM. However, in our Kitty algorithm, different channels in the Key cache are quantized with mixed precision (2-bit or 4-bit), which complicates the page loading process. To address this challenge, inspired by Wu et al. (2023) and Xia et al. (2024), we decompose each mixed-precision Key page into two tensors with unified 2-bit precision, as illustrated in Figure 3a.

We store all 2-bit channels and the lower two bits of boosted 4-bit channels in a dense tensor, `Tensor_2bits`, of shape (D, G). As shown in the lower part of Figure 3a, the higher two bits of boosted channels form a 2-bit sparse tensor. For memory efficiency, we compactly store only the nonzero channels of this sparse tensor, resulting in `Tensor_High_2bits` of shape (D_{boost}, G), where D_{boost} denotes the number of channels promoted to 4-bit precision.

To reconstruct the full tensor, we maintain an index tensor `Boost_IDX_uint8` with shape $(D,)$, defining a mapping $1, 2, \ldots, D \to 1, 2, \ldots, D_{\text{boost}}$ that maps each logical channel index to its physical offset in GPU memory. For channels not boosted to INT4 precision, the corresponding index is set to a sentinel value of $D_{\text{boost}} + 1$.

4.2 GPU Kernel Design for Page Dequantization.

Algorithm 1 Dequantize_KeyCache_Page()

shifts = [0, 2, 4, 6, 0, ...]
Input: Quantized cache $\mathbf{C}$, metadata $\mathbf{M}$, boosted channel number D_{boost}
Output: Dequantized key page $\mathbf{K}_{\text{fp16}} \in \mathbb{R}^{D \times T}$
scale, zero_point $\leftarrow$ LoadMeta($\mathbf{M}$)
boost_idx $\leftarrow$ Load($\mathbf{C}$)
boost_mask $\leftarrow$ (boost_idx $\leq D_{\text{boost}}$)
#$\mathbf{X}_{\text{low}}$ is an uint8 tensor with shape (D, T).
$\mathbf{X}_{\text{low}} \leftarrow$ LoadLowBits($\mathbf{C}$)
$\mathbf{X}_{\text{low}} \leftarrow (\mathbf{X}_{\text{low}} \gg \texttt{shifts}) \,\&\, 0x3$
#$\mathbf{X}_{\text{high}}$ is an uint8 tensor with shape (D, T).
$\mathbf{X}_{\text{high}} \leftarrow$ LoadHighBits($\mathbf{C}$, boost_idx, boost_mask)
$\mathbf{X}_{\text{high}} \leftarrow (\mathbf{X}_{\text{high}} \gg \texttt{shifts}) \,\&\, 0x3$
Combining and dequantization.
$\mathbf{X} \leftarrow \mathbf{X}_{\text{low}} \,|\, (\mathbf{X}_{\text{high}} \ll 2)$
$\mathbf{K}_{\text{fp16}} \leftarrow \mathbf{X} \odot \text{scale} + \text{zero\_point}$
return $\mathbf{K}_{\text{fp16}}$

Reconstructing an FP16 page from our quantized representation is non-trivial, particularly for the Key cache where each channel may have a different precision (2-bit or 4-bit). To address this challenge, our Kitty system leverages the previously introduced *Dense–Sparse Decomposition* and the

index tensor `Boost_IDX_uint8` to enable fully parallel page reconstruction on GPU.

Algorithm 1 presents the Triton-style pseudo code of this dequantization kernel, which reconstructs an FP16 tensor directly on-chip from quantized memory pages (Figure 3a). In line 8, an `uint8` tensor of shape (D, T) is loaded from HBM, where each byte encodes four consecutive 2-bit values. As a result, the same byte is logically reused by four times, introducing intentional redundancy in on-chip memory. In line 9, the lower two bits of each packed byte are extracted via a bit-shift and mask operation. For the higher two bits, line 10–11 conditionally fetches additional data from HBM according to the `boost_mask`. Only channels promoted to INT4 precision are read from the secondary tensor (`Tensor_High_2bits`), where the `boost_idx` tensor provides direct address mapping between the dense tensor and the compact boosted subspace. Finally, the kernel combines both low and high 2-bit components, applies per-channel scaling and zero-point correction, and reconstructs the full-precision $\mathbf{K}_{\text{fp16}}$ page on-chip. This design avoids divergent memory accesses and achieves efficient bit-level unpacking and parallel reconstruction for all channels. Notably, this dequantization process is fully fused within the downstream attention mechanism. By overlapping it with matrix multiplication, Kitty effectively hides the dequantization latency behind the dominant memory-bound operations, rendering the dequantization overhead negligible.

4.3 Kitty-Attention Execution Pipeline.

In this subsection, we present our novel execution pipeline for attention layers with quantized KV. The execution pipeline of traditional FP16 attention is straightforward, where the KV cache is first updated and then the attention output is computed. However, the situation becomes more complicated for the Kitty KV cache, as it requires coordinated management across multiple heterogeneous memory components, including the full-precision buffers (*Sink*, *Q-Buffer*, and *Local*) and the quantized pages stored in HBM. Each component serves a distinct purpose in balancing precision, memory efficiency, and temporal locality.

To manage these heterogeneous components efficiently, Kitty employs a three-stage execution pipeline. (1) inserting the new KV vectors into a static buffer (e.g. *Sink*, *Q-Buffer*, or *Local*); (2) loading the KV cache from our heterogeneous memory layout, reconstructing the FP16 KV cache with on-chip memory and computing the attention output; (3) and (optionally) performing quantization and packing of the KV vectors from the full precision buffers. To minimize the critical path latency, we employ a lazy-update strategy: step (1) performs only lightweight insertions, while the more compute-intensive maintenance operations are deferred to step (3). This ensures that the attention kernel can be dispatched immediately upon the arrival of new tokens. Furthermore, step (3) is triggered only once every G (e.g., 128) decoding steps, allowing its computational overhead to be effectively amortized across the entire generation sequence, thereby maintaining high inference throughput.

Step (1): Inserting the New KV Vectors in Full Precision. The newly generated KV vectors will be inserted into the KV cache before the attention computation. There might be multiple destinations for the KV vectors depending on the current state of each buffer. If the *Sink* is not full, the new KV will be directly inserted into it. Otherwise, the new Key tensor will be inserted into *Q-Buffer*, which is guaranteed to be not full due to Step (3), for the Key cache. For the Value cache, we also allocate a *Q-Buffer* to store the tensors to be quantized (in a large granularity rather than per-decoding-step to reduce the cost). Differently, the new Value tensor will be inserted into the *Local* buffer first if the *Sink* is full. Then, the oldest token in the *Local* will be evicted and inserted into *Q-Buffer* if *Local* is full. This is to guarantee that there is a fixed local window for Value cache.

Step (2): Attention Computation. We implemented the overall attention computation using two customized Triton GPU kernels (`qk_kernel` and `sv_kernel`) and one PyTorch operator (softmax). The attention logits are first computed with `qk_kernel`, where it loads the Query vector and the (quantized) Key cache from GPU memory and compute the matrix multiplication. Then, the attention score (full precision) is computed with PyTorch softmax operator. Finally, the attention scores and the (quantized) Value cache are multiplied with `sv_kernel`. We leave the further optimization of these kernels (e.g., fusing these kernels into one (Dao et al., 2022)) for future work.

Step (3): Quantization and Packing After the attention computation, quantization and packing process will be launched once the *Q-Buffer* is full. During this process, all the KV vectors in the *Q-Buffer* will be quantized with our customized Triton kernel and packed into one page (described in Figure 3). Given that the size of *Q-Buffer* is G and at most one vector can be inserted to *Q-Buffer* for each decoding step, this quantization process can be launched at most once every G decoding steps. In this way, the quantization overhead is efficiently **amortized** and becomes negligible.

5 Experimental Results

5.1 Setup

Evaluated Models. We evaluate the accuracy of *Kitty* on two families of open-source reasoning models: Qwen3 (Yang et al., 2025) (8B, 14B, 32B) and LLaMA

3 (Dubey et al., 2024) (8B, 70B).

Evaluation Datasets. The accuracy impact was evaluated mainly on four reasoning benchmarks: (1) two mathematical reasoning datasets: GSM8K (Cobbe et al., 2021) and MATH-Algebra (Hendrycks et al., 2021); (2) a code generation benchmark HumanEval (Chen et al., 2021); and (3) a graduate-level science question dataset GPQA-Diamond (Rein et al., 2023). We also conducted extended evaluations on advanced math problems AIME24 (math ai, 2024) and AIME25 (math ai, 2025). For HumanEval, we report functional correctness using pass@1 as the primary metric, while for the other benchmarks we report accuracy by comparing the extracted model outputs against ground-truth labels, following standard practice.

Evaluation Frameworks. We conduct end-to-end accuracy evaluation using our simulation framework. Meanwhile, we perform system evaluations using our novel inference engine described in Section 4. Downstream task evaluations are performed via *lm-evaluation-harness* (EleutherAI Team, 2025), a widely adopted framework for standardized benchmarking of large language models. To better capture accuracy under long-context generation, we adopt the chain-of-thought (CoT) variants of evaluation prompts. Following common practice, we evaluate GSM8K with 8-shot prompts, MATH with 4-shot prompts, and GPQA with 5-shot prompts.

Evaluation Configurations. All evaluations are conducted using PyTorch 2.4.1 with CUDA 12.1 and FlashAttention `2.7.4.post1` (Dao et al., 2022). We evaluate the accuracy of Qwen3-8B and LLaMA3-8B on NVIDIA A100 GPUs (NVIDIA, 2020), and the larger models on NVIDIA H100 GPUs (NVIDIA, 2023). Following common practice, we enable stochastic sampling during inference, setting the *temperature* to 0.6, *top_p* to 0.95, and *top_k* to 20. During accuracy evaluation, the maximum number of generated tokens is limited to 4096. For AIME tasks, we extend the maximum generation length to 32768 tokens to accommodate the long-chain reasoning required by such problems. To ensure the stability of our results, each experiment is repeated anywhere from 3 up to 10 times, and we report both the average accuracy and the maximum observed deviation.

5.2 Accuracy Results: Accuracy Recovery

In order to verify the effectiveness of the quantization scheme presented in Section 3, we conducted comprehensive accuracy comparisons between our methods and existing work. The overall results for Qwen3 models and LLaMA3 models are summarized in Table 3. "Average" here is the mean across benchmarks, and "Difference" is the difference of average accuracy versus the FP16 baseline (K16V16). For each model, we evaluate the inference accuracy of the baseline KV cache (K16V16), variances of KIVI quantization algorithm, and variances of our Kitty algorithm. The K16V16 here is the original implementation of HuggingFace Tranformers with FP16 KV cache. KIVI-K2V2 uses the KIVI (Liu et al., 2024) algorithm to quantize the KV cache into 4 and 2 bits. For better ablations, KIVI-K2V2* is the extended version of KIVI-K2V2 where the initial tokens are preserved in full precision. The details of our Kitty variants are described in Section 3, where a small fraction (12.5%, 25%) of the key-cache channels are quantized to 4-bit precision while the rest of the channels are quantized to 2-bit precision, results in Kitty and Kitty-Pro. Below we summarize the main observations and their practical implications.

KIVI-K2V2. The KIVI algorithm fails to preserve model accuracy under aggressive 2-bit KV cache quantization. As shown in Table 3, KIVI-K2V2 consistently yields significantly lower accuracy than the FP16 baseline (**K16V16**) across all evaluated tasks. This degradation becomes particularly evident on reasoning benchmarks such as GSM8K and MATH-Algebra.

KIVI-K2V2*. KIVI-K2V2* demonstrates substantial accuracy recovery compared to KIVI-K2V2, effectively narrowing the degradation gap across multiple models. This improvement aligns with our findings in Section 3.1, highlighting the importance of preserving the initial tokens in full precision. However, despite the partial recovery, KIVI-K2V2* still lags behind the FP16 baseline, especially on complex reasoning tasks such as MATH-Algebra.

Kitty and Kitty-Pro. Our proposed Kitty with channel-wise precision boost strategy effectively bridges the remaining accuracy gap left. As shown in Table 3, Kitty consistently recovers accuracy across all evaluated models, surpassing KIVI-K2V2* by 5.17 on Qwen3-8B in terms of average accuracy. The enhanced variant, Kitty-Pro, further increases the boosted channel ratio (from 12.5% to 25%), achieving near-parity or even slight improvements over the FP16 baseline on several models such as Qwen3-14B and LLaMA3.3-70B-Instruct. These results demonstrate that incorporating channel-wise sensitivity is a principled and scalable approach to restoring accuracy under aggressive 2-bit quantization.

Component Isolation. To provide clear ablation of all components of our quantization scheme (Figure 1), we isolate the impact of each component, especially showing the isolated accuracy benefits of our *channel-wise precision boost* strategy. In Table 4, we explicitly show the **incremental** accuracy gains of: KIVI-2 (Baseline) $\rightarrow$ + FP16 Sink Tokens $\rightarrow$ + Channel-wise Precision Boost (10% / 20%). While preserving FP16 Sink Tokens provides an initial recovery from a highly degraded baseline, our *channel-wise precision boost* strategy achieves substantial further gains

Table 3. Benchmark results of different KV cache quantization methods.

Model / Method		GSM8K	MATH ALGEBRA	HUMAN EVAL	GPQA DIAMOND	Average	Drop
Qwen3-8B	**K16V16**	**94.79**$\pm$0.71	**88.26**$\pm$0.45	**84.82**$\pm$3.72	**40.71**$\pm$2.96	**77.15**	-
	KIVI-K2V2	89.13$\pm$0.51	47.29$\pm$0.20	76.89$\pm$3.11	32.24$\pm$3.16	61.39	-15.76
	KIVI-K2V2*	89.71$\pm$0.63	74.92$\pm$1.83	78.54$\pm$2.56	36.02$\pm$3.27	69.80	-7.35
	Kitty	93.61$\pm$0.58	85.12$\pm$1.54	81.77$\pm$1.89	39.39$\pm$4.18	74.97	-2.18
	Kitty-Pro	94.34$\pm$0.48	88.12$\pm$1.26	81.34$\pm$3.41	40.92$\pm$5.00	76.18	-0.97
Qwen3-14B	**K16V16**	**94.69**$\pm$0.45	**90.68**$\pm$0.14	**86.18**$\pm$2.03	**47.62**$\pm$3.74	**79.79**	-
	KIVI-K2V2	75.82$\pm$1.97	83.66$\pm$1.01	83.74$\pm$0.41	41.50$\pm$0.34	71.18	-8.61
	KIVI-K2V2*	89.56$\pm$0.94	83.74$\pm$0.59	85.98$\pm$1.83	45.24$\pm$2.89	76.13	-3.66
	Kitty	94.67$\pm$0.94	90.31$\pm$0.93	88.21$\pm$2.24	47.11$\pm$4.25	80.08	0.29
	Kitty-Pro	94.90$\pm$0.35	90.54$\pm$0.39	86.38$\pm$1.63	45.92$\pm$2.55	79.44	-0.35
Qwen3-32B	**K16V16**	**91.74**$\pm$0.99	**84.84**$\pm$0.93	**85.98**$\pm$1.22	**48.81**$\pm$1.87	**77.84**	-
	KIVI-K2V2	88.17$\pm$0.38	59.70$\pm$0.81	84.76$\pm$0.61	44.22$\pm$1.19	69.21	-8.63
	KIVI-K2V2*	89.31$\pm$0.91	74.92$\pm$0.45	85.37$\pm$2.44	48.98$\pm$2.55	74.65	-3.19
	Kitty	91.56$\pm$0.58	83.80$\pm$0.62	86.28$\pm$5.79	47.45$\pm$1.02	77.27	-0.57
	Kitty-Pro	90.60$\pm$0.91	83.80$\pm$1.29	88.21$\pm$0.41	51.53$\pm$0.51	78.54	0.70
LLaMA3.1-8B-Instruct	**K16V16**	**76.75**$\pm$1.16	**47.15**$\pm$0.48	**63.96**$\pm$3.11	**26.94**$\pm$5.00	**53.70**	-
	KIVI-K2V2	63.58$\pm$0.63	31.45$\pm$0.62	55.30$\pm$8.72	23.88$\pm$8.57	43.55	-10.15
	KIVI-K2V2*	71.04$\pm$0.83	44.12$\pm$0.62	56.71$\pm$4.88	23.67$\pm$3.88	48.89	-4.81
	Kitty	75.99$\pm$0.96	45.97$\pm$1.57	60.00$\pm$5.85	25.41$\pm$3.47	51.84	-1.86
	Kitty-Pro	75.51$\pm$1.06	47.37$\pm$0.90	61.65$\pm$5.43	25.82$\pm$3.78	52.59	-1.11
LLaMA3.3-70B-Instruct	**K16V16**	**94.92**$\pm$0.30	**71.58**$\pm$1.12	**83.13**$\pm$2.03	**45.92**$\pm$2.55	**73.89**	-
	KIVI-K2V2	93.91$\pm$0.56	66.05$\pm$0.76	79.07$\pm$2.24	45.07$\pm$5.44	71.03	-2.86
	KIVI-K2V2*	94.77$\pm$0.30	69.64$\pm$0.81	82.72$\pm$2.85	47.11$\pm$2.38	73.56	-0.33
	Kitty	95.05$\pm$0.63	70.35$\pm$0.59	83.13$\pm$1.02	45.58$\pm$3.23	73.53	-0.36
	Kitty-Pro	95.00$\pm$0.23	70.77$\pm$0.59	82.72$\pm$0.81	46.94$\pm$3.06	73.86	-0.03

Notes. `K16V16` denotes 16-bit precision for the KV cache, while `KIVI-K2V2` applies 2-bit quantization using the KIVI (Liu et al., 2024) algorithm; an asterisk (*) indicates that the first 32 tokens remain in full precision. `Kitty` and `Kitty-Pro` preserve 12.5% and 25% of channels in INT4. The maximum generation length is 4096 tokens.

on top of an already optimized state. As illustrated in Table 4, while the preservation of sink tokens harvests the 'low-hanging fruit' of initial accuracy recovery, our strategy consistently narrows the residual accuracy gap. This task becomes increasingly non-trivial as the model approaches its full-precision ceiling, where the marginal utility of additional optimizations typically diminishes.

Summary. Our results suggest that *channel-wise precision boost* is an effective strategy to mitigate accuracy loss from aggressive KV quantization. Kitty-Pro achieves near-parity with FP16 accuracy across multiple benchmarks and models, while retaining the memory benefits of low-bit KV cache.

5.3 Extended Results on Longer Context Length

In Table 3, the maximum generation length is set to 4,096 tokens. To examine our method's robustness under longer-context reasoning, we further evaluated AIME24 and AIME25 with a maximum generation length of 32,768 tokens. As shown in Table 5, the degradation of 2-bit quantization becomes more pronounced, with KIVI-K2V2 suffering an average accuracy drop of around 13 points compared to the FP16 baseline. While KIVI-K2V2* alleviates part of the degradation, it still lags behind the FP16 baseline by 6–8 points. In contrast, our proposed Kitty achieves substantial recovery, narrowing the average gap to only 3–4 points across all models. These results demonstrate that Kitty maintains more stable accuracy even at 32k-token context lengths, showing strong robustness under long-context reasoning settings.

5.4 Ablation Study on Channel-wise Precision Boost

In Section 5.2, we demonstrate the effectiveness of Kitty and Kitty-Pro. In this section, we conduct ablation study by more comprehensive investigating the impact of using

Table 4. Isolated accuracy improvements of applying components in Figure 1 to KIVI-K2V2 baseline. Each column from left to right represents the incremental gain achieved by successively integrating each optimization component. This table is derived from the average accuracy shown in Table 3.

Model	+ FP16 Sinks	+ 10% INT4 Ch.	+ Another 10% Ch.
Qwen3-8B	+8.41	+5.17	+1.21
Qwen3-14B	+4.95	+3.95	-0.64
Qwen3-32B	+5.44	+2.62	+1.27
LLaMA3-8B	+5.34	+2.95	+0.75
LLaMA3-70B	+2.53	-0.03	+0.33

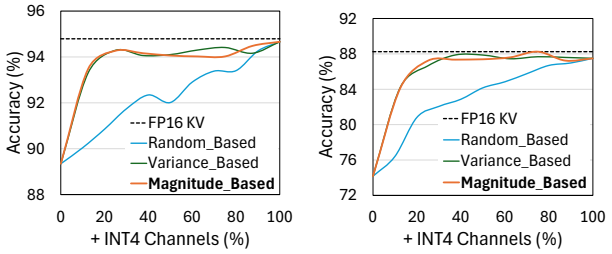
Table 5. Qwen3-8B results on AIME24 and AIME25 (max generation length: 32k tokens). KIVI-KV2* here denotes the variant of KIVI where the KVs of initial tokens are not quantized.

Model / Method		AIME24	AIME25	Avg.
Qwen-8B	**KV16**	**71.67**$\pm_{15.00}$	**66.00**$\pm_{7.33}$	**68.84**
	KIVI-KV2	57.00$\pm_{7.00}$	52.33$\pm_{9.00}$	54.67
	KIVI-KV2*	67.67$\pm_{9.00}$	57.67$\pm_{9.00}$	62.67
	Kitty	70.67$\pm_{7.33}$	59.67$\pm_{10.33}$	65.17
Qwen3-14B	**KV16**	**80.67**$\pm_{7.33}$	**69.33**$\pm_{7.33}$	**75.00**
	KIVI-KV2	69.00$\pm_{9.00}$	55.33$\pm_{8.67}$	62.17
	KIVI-KV2*	74.67$\pm_{8.00}$	61.33$\pm_{8.67}$	68.00
	Kitty	75.67$\pm_{4.33}$	67.33$\pm_{9.33}$	71.50
Qwen3-32B	**KV16**	**81.67**$\pm_{5.00}$	**72.59**$\pm_{7.41}$	**77.13**
	KIVI-KV2	73.00$\pm_{9.67}$	57.41$\pm_{9.26}$	65.21
	KIVI-KV2*	79.00$\pm_{9.00}$	59.05$\pm_{12.38}$	69.03
	Kitty	79.67$\pm_{9.67}$	69.26$\pm_{9.26}$	74.47

different *channel boost rate*[4]. Figure 4 shows the accuracy trends on Qwen3-8B when varying the fraction of key-cache channels boosted to INT4. A flat baselines are also plotted for K16V16 baseline (upper dashed line).

Monotonic Accuracy Improvements. As shown in Figures 4, accuracy on GSM8K and MATH_Algebra improves almost monotonically as the *channel boost rate* increases. Notably, accuracy is effectively recovered once 25% of channels are boosted to 4-bit precision. This mirrors the table-level averages in Table 3, where Kitty-Pro reaches parity. Overall, we find that accuracy improves consistently with higher *channel boost rates* across most tasks. Minor accuracy fluctuations here are attributable to inherent statistical variance in the evaluation process rather than systematic regression, consistent with the standard deviation in Table 3.

[4]We use the term *channel boost rate* to denote the fraction of channels boosted from 2-bit to 4-bit precision.



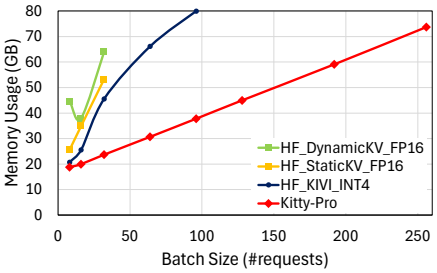
(a) GSM8K (b) MATH-Algebra

Figure 4. Accuracy recovery with *channel-wise precision boost* on Qwen3-8B. Similar trends are observed on other tasks.

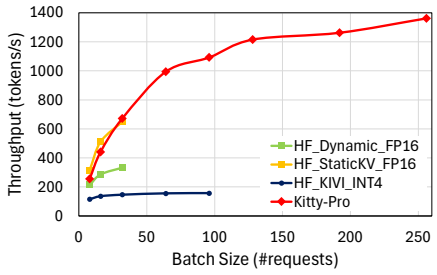
Necessity of Selection Heuristic. To examine whether heuristics are indeed necessary for channel-wise precision boost, we include a random selection baseline, where each channel is assigned a random importance score. By comparing the accuracy curves in Figures 4, we verified that while randomly boost some channels can improve accuracy, heuristic-guided *channel-wise precision boost* yields substantially greater benefits. To provide an alternative heuristic to magnitude-based selection, we also provide a side-by-side comparison between Magnitude-based and Variance-based heuristics in Figure 4. We also notice that on-par accuracy improvements are achieved for magnitude-based and variance-based selection. Given that magnitude-based introduce lower runtime cost in computing the score for each channel, we mainly discuss this heuristic in this paper.

5.5 System Results: Inference Efficiency

To evaluate system-level efficiency of Kitty, we performs end-to-end Qwen3-8B inference using our prototype inference engine presented in Section 4. A batch of example prompts are fed to the inference system and we let the system generate tokens until it hits the predefined maximum sequence length (8192). We also include two Hugging Face baselines here to demonstrate the system-level benefits of our KV quantization method. *HF_Dynamic_FP16* here denotes the default configuration of Hugging Face transformers library, where the KV cache grows dynamically as more KV vectors are cached in the system. *HF_Static_FP16* is a more runtime-efficient KV cache implementation, where the KV cache is pre-allocated before the token generation. Furthermore, we also evaluated the INT4 KV cache implemented by Hugging Face, denoted as HF_KIVI_INT4, which is a re-implementation of KIVI (Liu et al., 2024). The prompts are around 100 tokens so the inference time is mainly dominated by the decoding phase. For each KV cache implementations, we increase the batch size until out-of-memory, and report the peak memory usage and token generation throughput under different batch sizes. The hardware we used is a single NVIDIA A100 (NVIDIA, 2020)



(a) GPU Memory Usage.



(b) Inference Throughput.

Figure 5. Memory usage and throughput comparison on Qwen3-8B generating 8192 tokens. Kitty can achieve higher throughput by enabling larger batch sizes.

GPU (80GB).

Runtime and Memory Overhead Analysis. Our profiling results on Qwen3-8B (batch size 32) demonstrate that a typical decoding step takes 47 ms. Within each decoding step, the raw quantization overhead for Keys and Values is 4.4 ms and 1.2 ms, respectively (∼12% of total latency). Additionally, computing saliency scores for each channel incurs a 4.6 ms overhead (∼10% of total latency). While the aggregate cost of channel selection and KV quantization is 10.2 ms, this operation is triggered only once every G (e.g., 128) steps. When amortized across the generation sequence, the effective per-step overhead is reduced to a negligible 0.08 ms (0.17%). This confirms that our dynamic selection mechanism—while sophisticated—imposes minimal impact on the overall inference throughput.

In our practical implementation, the total HBM footprint encompasses: (i) the primary quantized payload, (ii) the FP16 sink and local cache buffers, and (iii) the essential quantization metadata. Accounting for these auxiliary overheads, the effective average bit-width can be modeled as:

$$\text{Avg. Bit-width} = \frac{2200}{N} + 2.357, \tag{7}$$

where N denotes the sequence length. As N increases, the fixed costs of sink tokens and metadata are rapidly amortized. For instance, at a 32K sequence length, the effective bit-width converges to ≈ 2.44 bits—achieving a $6.6\times$ compression ratio that significantly outperforms standard 4-bit baselines while nearing the theoretical 2-bit limit.

Throughput Improvements. Our target deployment regime is *throughput-bound, long-context serving*, where HBM capacity—not latency—is the primary bottleneck. As shown in Figure 5, with the same memory budget, Kitty-Pro enables $8\times$ batch sizes and achieves $2.1\times \rightarrow 4.1\times$ higher inference throughput compared to the FP16 baseline. This throughput leap stems from our superior memory compression, which enables the system to amortize the fixed HBM access overhead across a significantly larger number of concurrent requests. Consequently, Kitty effectively translates its memory efficiency into increased aggregate serving capacity, maximizing GPU utilization under strict memory constraints.

The throughput gap between HF_KIVI and Kitty as illustrated in Figure 5b is primarily driven by two architectural optimizations. First, Kitty employs on-the-fly dequantization fused directly into the attention kernels. This integration eliminates redundant intermediate memory traffic, whereas the baseline relies on discrete operators that incur heavy Global Memory I/O overhead. Second, while the baseline performs quantization at every decoding step, Kitty effectively amortizes the dynamic quantization costs through our periodic update strategy (Section 4.3). Furthermore, our current implementation utilizes Triton (Tillet et al., 2019) as a high-level abstraction for proof-of-concept. The inference efficiency can be further improved if using more low-level programming language, e.g., CUDA, and enabling more fine-grained optimizations. We leave these engineering refinements as future work.

6 Conclusions

This paper identified that 4-bit KV cache quantization largely preserves accuracy, while existing 2-bit quantization method leads to substantial degradation on reasoning-intensive tasks. Preserving initial tokens in full precision helps but does not fully close the gap with FP16. Our channel-wise precision boost method further narrows this gap: boosting only 12.5%-25% of key-cache channels to higher precision is often sufficient to recover most of the accuracy loss. Based on this innovation, we propose an accurate and memory-efficient quantization framework, *Kitty*. Furthermore, we propose corresponding system designs to support the end-to-end inference with Kitty. The end-to-end evaluations show that our 2-bit inference system can significantly reduce the GPU memory consumption and enable $8\times$ larger batch sizes during inference, which results in $2.1\times \rightarrow 4.1\times$ higher inference throughput compared to the FP16 baseline with same memory budget.

REFERENCES

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms, 2024. URL https://arxiv.org/abs/2404.00456.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Du, D., Cao, S., Cheng, J., Mai, L., Cao, T., and Yang, M. Bitdecoding: Unlocking tensor cores for long-context llms with low-bit kv cache, 2025. URL https://arxiv.org/abs/2503.18773.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

EleutherAI Team. Lm evaluation harness. https://github.com/EleutherAI/lm-evaluation-harness, 2025. GitHub repository, Accessed: 2025-09-17.

Gale, T., Zaharia, M., Young, C., and Elsen, E. Sparse gpu kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Hugging Face Team. Transformers library. https://github.com/huggingface/transformers, 2025. GitHub repository, Accessed: 2025-09-17.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Li, X., Xing, Z., Li, Y., Qu, L., Zhen, H.-L., Liu, W., Yao, Y., Pan, S. J., and Yuan, M. Kvtuner: Sensitivity-aware layer-wise mixed-precision kv cache quantization for efficient and nearly lossless llm inference, 2025. URL https://arxiv.org/abs/2502.04420.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Liu, R., Sun, Y., Zhang, M., Bai, H., Yu, X., Yu, T., Yuan, C., and Hou, L. Quantization hurts reasoning? an empirical study on quantized reasoning models, 2025. URL https://arxiv.org/abs/2504.04823.

Liu, Z., Yuan, J., Jin, H., Zhong, S. H., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

math ai. Aime 2024 benchmark. https://huggingface.co/datasets/math-ai/aime24, 2024. American Invitational Mathematics Examination 2024 problems curated as a reasoning benchmark.

math ai. Aime 2025 benchmark. https://huggingface.co/datasets/math-ai/aime25, 2025. American Invitational Mathematics Examination 2025 problems curated as a reasoning benchmark.

NVIDIA. Nvidia a100 tensor core gpu architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020. Accessed: 2025-09-17.

NVIDIA. Nvidia h100 tensor core gpu architecture. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c, 2023. Accessed: 2025-09-17.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof qa benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Sharma, A., Ding, H., Li, J., Dani, N., and Zhang, M. Minikv: Pushing the limits of llm inference via 2-bit layer-discriminative kv cache. *arXiv preprint arXiv:2411.18077*, 2024.

Tillet, P., Kung, H. T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

Wu, X., Xia, H., Youn, S., Zheng, Z., Chen, S., Bakhtiari, A., Wyatt, M., Aminabadi, R. Y., He, Y., Ruwase, O., Song, L., and Yao, Z. Zeroquant(4+2): Redefining llms quantization with a new fp6-centric strategy for diverse generative tasks, 2023. URL https://arxiv.org/abs/2312.08583.

Xia, H., Zheng, Z., Li, Y., Zhuang, D., Zhou, Z., Qiu, X., Li, Y., Lin, W., and Song, S. L. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Proc. VLDB Endow.*, 17(2):211–224, October 2023. ISSN 2150-8097. doi: 10.14778/3626292.3626303. URL https://doi.org/10.14778/3626292.3626303.

Xia, H., Zheng, Z., Wu, X., Chen, S., Yao, Z., Youn, S., Bakhtiari, A., Wyatt, M., Zhuang, D., Zhou, Z., Ruwase, O., He, Y., and Song, S. L. Quant-LLM: Accelerating the serving of large language models via FP6-Centric Algorithm-System Co-Design on modern GPUs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 699–713, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL https://www.usenix.org/conference/atc24/presentation/xia.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024a. URL https://arxiv.org/abs/2211.10438.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks, 2024b. URL https://arxiv.org/abs/2309.17453.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

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

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024b. Curran Associates Inc.

Zhao, Y., Lin, C.-Y., Zhu, K., Ye, Z., Chen, L., Zheng, S., Ceze, L., Krishnamurthy, A., Chen, T., and Kasikci,

B. Atom: Low-bit quantization for efficient and accurate llm serving. In Gibbons, P., Pekhimenko, G., and Sa, C. D. (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 196–209, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/5edb57c05c81d04beb716ef1d542fe9e-Paper-Conference.pdf.

Zheng, Z., Song, X., and Liu, C. Mixllm: Llm quantization with global mixed-precision between output-features and highly-efficient system design, 2024. URL https://arxiv.org/abs/2412.14590.