TriAxialKV: Toward Extreme Low-Precision KV-Cache Quantization for Agentic Inference Tasks

Hanzhang Shen[1] **Haoran Wu**[1] **Yiren Zhao**[2] **Robert Mullins**[1,*]
[1] University of Cambridge [2] Imperial College London

Abstract

Agentic workloads have emerged as a major workload for LLM inference. They differ significantly from chat-only workloads, requiring long-context processing, the ability to handle multimodal inputs, and structured multi-turn interactions with tool calling capabilities. As a result, their context exhibits structure that can carry different importance along three key axes: temporal recency to the current turn, modality such as text or image tokens, and semantic role such as user queries, tool calls, observations, or reasoning. These axes capture distinct token behaviors and lead to different sensitivities to KV-cache compression. However, existing KV-cache quantization methods are typically homogeneous or exploit only heterogeneity on a single dimension, such as temporal proximity or modality, overlooking the interactions among them. To this end, we introduce **TriAxialKV**, a novel mixed-precision KV-cache quantization scheme that assigns each token a triaxial tag, calibrates per-tag sensitivity, and allocates INT2/INT4 bit-widths under a fixed memory budget. We implement TriAxialKV as an end-to-end serving system, comprising calibration, mixed-precision quantization and memory management, and custom fused Triton decode kernels. When using Qwen3-VL-32B-Thinking as a computer-use agent operating the OSWorld, TriAxialKV matches the accuracy of SGLang with BF16 KV cache while supporting 4.5× KV cache capacity and achieving 30% higher end-to-end throughput, when running on real GPU systems.

1 Introduction

Large language models (LLMs) have rapidly evolved from conversational assistants [43, 54, 6] to agentic applications such as web search [9, 18], computer use [2, 1], and code generation [10, 28, 35], where the model autonomously plans, invokes tools, consumes multimodal inputs, and reasons over feedback across many turns. These workloads require the model to attend over very long, multi-turn contexts, causing the key–value (KV) cache to grow rapidly and become a dominant memory bottleneck in LLM inference. As the KV cache expands, it can quickly saturate GPU memory, thereby limiting the achievable batch size and throughput [44, 14]. For instance, a 100K-token context with Qwen3-VL-32B [5] requires approximately 24 GiB of KV-cache memory at BF16 precision, even for a single request. Combined with model weights and runtime buffers, such large KV caches leave substantially less GPU memory for concurrent requests, limiting batch size and serving throughput.

KV-cache compression is a promising approach to improve agentic serving efficiency. A wide range of methods have been proposed, including uniform low-bit quantization [21, 4], asymmetric quantization [27, 38], token-eviction strategies that exploit attention sparsity [52, 53], and adaptive bit-width allocation along the layer depth or generation axis [7, 26]. However, these methods primarily adapt compression according to quantization statistics, attention-based importance, or individual structural properties of the KV cache, without fully accounting for the heterogeneous nature of agentic workloads. In particular, agentic contexts exhibit substantial heterogeneity along three dimensions:

*Corresponding author: Robert.Mullins@cl.cam.ac.uk

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

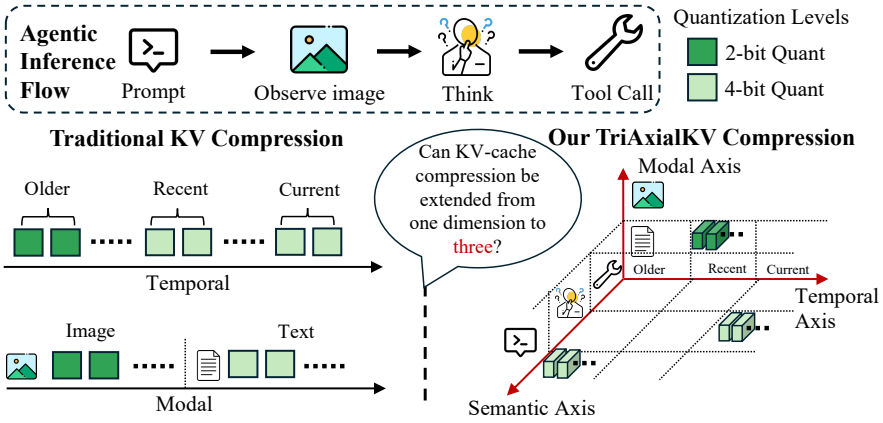


Figure 1: Comparison of single-axis KV-cache compression methods, including PM-KVQ [26] for temporal compression and VL-Cache [40] for modality-aware compression, with TriAxialKV, which holistically compresses the KV cache by jointly considering temporal, modal, and semantic axes.

tokens can (i) originate from different *modalities* (e.g., text, image), (ii) arrive at different *times* in the conversation (system prompts, early user turns, recent tool outputs), and (iii) play different *semantic* roles (instructions, reasoning, tool-call arguments, tool responses).

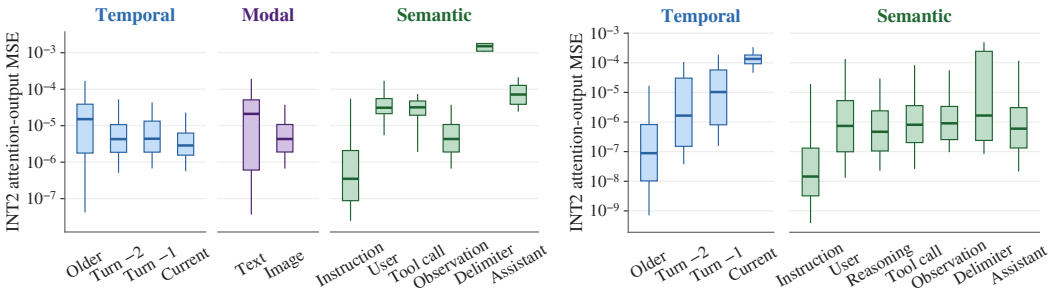
Recent works have begun to exploit individual dimensions for KV-cache compression. ThinKV [34] considers the semantic role of tokens, VL-Cache [40] and LOOK-M [41] classify tokens based on modality, and PM-KVQ [26] categorizes reasoning tokens according to temporal recency. However, as illustrated in Figure 1, they *treat these dimensions in isolation rather than jointly modeling the temporal, modal, and semantic structure of agentic traces*. Consequently, existing compression policies cannot distinguish tokens that share the same tag along one axis but differ in quantization sensitivity along the others, leading to accuracy degradation under aggressive compression.

To address this gap, we propose **TriAxialKV**, a mixed-precision KV-cache quantization framework that assigns each token a triaxial tag based on its temporal recency, modality, and semantic role. By calibrating the quantization sensitivity of each tag, TriAxialKV allocates INT2/INT4 precision under a memory budget, enabling aggressive KV-cache compression while preserving model accuracy. In summary, our main contributions are:

- By profiling function-calling and computer-use prefills, we show that KV quantization sensitivity is structured along three complementary axes: *temporal recency*, *modality*, and *semantic role*. To our knowledge, this is the *first* work to exploit all three jointly for KV-cache quantization.
- Building on this finding, we develop a *taxonomy-driven calibration and allocation framework*: a chat-template-based tagger that labels every prefill token in a single pass without any model inference, a calibration procedure that measures per-tag attention-output distortion from real prefill captures, and a per-tag bit-width allocator that solves the INT2/INT4 assignment problem.
- We deliver an *end-to-end serving system* that integrates the tagger, allocator, and mixed-precision memory pool with fused Triton decode kernels into SGLang [55]. On Qwen3-VL-32B [5] running OSWorld [46], our system provides *4.5× the KV-cache capacity* and serves *1.3× the end-to-end throughput* of the BF16 baseline on an NVIDIA B200 GPU while matching its accuracy. On BFCL Memory [31], it stays within 2% of BF16 accuracy across four models, while KIVI [27] and SGLang FP4 lose up to 6.0% and 4.7%.

2 Related Work

KV cache compression To reduce the memory footprint of long-context inference and enable larger batch sizes, prior work has extensively explored KV cache quantization. KIVI [27] employs two-bit per-channel quantization for keys and per-token quantization for values to account for their distributions. KVQuant [20] enables sub-4-bit precision through pre-RoPE key quantization, non-



(a) Qwen3-VL-8B, OSWorld

(b) Qwen3-32B, BFCL

Figure 2: INT2 quantization sensitivity of 32-token same-tag groups along each tag axis. Each group's KV is quantized to INT2 alone, with all other KV entries kept in BF16.

uniform datatypes, and outlier isolation. QServe [25] co-designs W4A8KV4 quantization with an efficient serving system to translate memory savings into throughput improvements. OSCAR [57] calibrates per-layer rotations and clipping thresholds offline from attention-aware KV covariance, enabling 2-bit KV quantization with a small BF16 sink and recent window.

Other approaches exploit variations in KV cache sensitivity to allocate precision non-uniformly. ZipCache [19] identifies salient tokens via channel-separable scoring to enable aggressive compression of less-important tokens. PM-KVQ [26] employs progressive mixed-precision quantization to reduce the bit-width of KV cache blocks. ThinKV [34] adapts quantization and eviction to the importance of different reasoning thoughts, whereas ChanMix [24] allocates precision according to channel sensitivity. While these methods demonstrate the benefits of sensitivity-aware quantization, our work jointly exploits temporal recency, modality, and semantic role to guide precision allocation for agentic workloads. A complementary line of work explores token eviction for KV-cache compression [52, 45, 17, 23, 8, 16, 53]. Token eviction is complementary to our approach, as it reduces the number of retained tokens, whereas our method reduces the precision of retained KV entries.

Efficient LLM serving systems A substantial body of work has addressed system-level challenges in LLM serving. Orca [50] introduces iteration-level scheduling and selective batching to improve GPU utilization. Building on iteration-level batching, Sarathi-Serve [3] introduces chunked prefills and stall-free scheduling to balance throughput and tail latency. FlashAttention [12, 11, 36, 51] reduces memory traffic and optimizes GPU kernel execution for efficient attention computation. vLLM [22] uses PagedAttention to manage KV cache in non-contiguous blocks and reduce memory fragmentation. SGLang [55] employs a radix tree to organize KV cache and enable prefix reuse among requests. Another line of work explores prefill–decode disaggregation, as exemplified by DistServe [56] and Splitwise [30]. Mooncake [32] further incorporates disaggregated KV cache storage and cache-aware scheduling. Our work complements these system-level optimizations by reducing KV cache memory consumption through sensitivity-aware quantization. We integrate our approach into SGLang to demonstrate end-to-end serving throughput improvements.

3 Method

An agentic prefill request consists of heterogeneous segments, such as the system prompt, user queries, tool calls, and tool outputs. To measure their quantization sensitivity, we tag every prefill token by its *temporal recency*, *modality*, and *semantic role*, and split the tokens sharing all three tags into groups of 32. We quantize one group at a time to INT2, keeping all else in BF16, and record the attention-output mean-squared error (MSE) at a middle layer. As shown in Figure 2, semantic role separates segments on both workloads, with instruction tokens one to two orders of magnitude less sensitive than most other roles. Temporal recency matters more on BFCL, where sensitivity rises monotonically by over two orders of magnitude toward the current turn. Text and image tokens differ by only about $5\times$, but images make up 84% of OSWorld (computer-use) prefill tokens, so assigning them INT2 frees budget to keep more sensitive tokens at INT4.

This structure enables a *sensitivity-aware allocation* that assigns a bit-width to each tag combination, which we call a *triaxial tag*, subject to an average bit-width budget. As shown in Figure 3, our

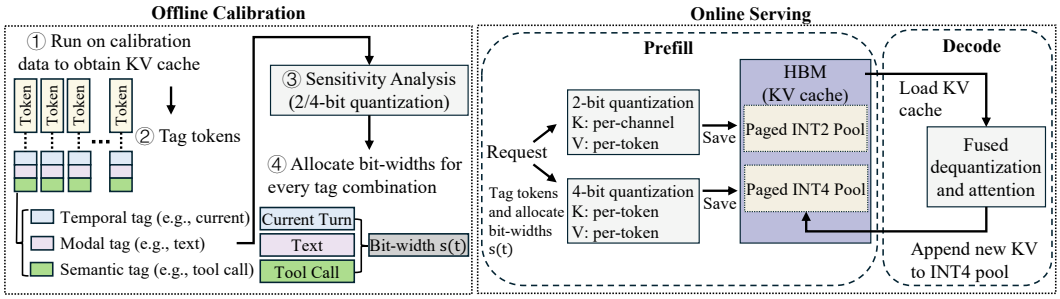


Figure 3: Overview of our TriAxialKV compression flow. During prefill, KV entries are quantized and stored in INT2/INT4 pools based on the bit-width assigned to the token's triaxial tag.

Table 1: Semantic axis values in our tag taxonomy.

Semantic Tag	Description
`inst`	system prompt and tool schemas
`user`	user-message text (excluding tool outputs)
`assistant`	assistant text outside reasoning and tool-call blocks
`reasoning`	chain of thought inside `<think>` blocks
`tool_call`	tool invocations inside `<tool_call>` blocks
`obs`	environment feedback (tool outputs, screenshots, etc.)
`delim`	role markers and the open/close tokens of the regions above

system has two stages. First, an offline calibration stage measures the attention-output distortion of quantizing each triaxial tag to INT2 and INT4, and solves for a bit-width allocation under the budget. Second, at serving time, the system tags each prefill token and looks up the bit-width of its triaxial tag. We maintain separate paged INT2 and INT4 memory pools, and implement a fused Triton decode kernel that runs Flash-Decoding [13] with on-the-fly dequantization of both bit-widths.

3.1 Triaxial Tagging and Bit-Width Allocation

Tag taxonomy. Our *tag taxonomy* gives each prefill token t one value on each of three axes: temporal, modal, and semantic. The tuple of these values is its triaxial tag $s(t)$ in the joint tag space

$$\mathcal{S} = \mathcal{A}_{\text{temporal}} \times \mathcal{A}_{\text{modal}} \times \mathcal{A}_{\text{semantic}}, \tag{1}$$

where each axis captures a distinct property of the segment.

Temporal. $\mathcal{A}_{\text{temporal}} = \{\texttt{older}, \texttt{turn_m2}, \texttt{turn_m1}, \texttt{current}\}$. We group tokens by their distance from the most recent turn, where a turn spans from one user message up to, but not including, the next. `current` covers the most recent turn, `turn_m1` and `turn_m2` cover the two turns before it, and `older` covers everything earlier. Because attention to a turn tends to change as the conversation moves on, tokens from different turns can differ in quantization sensitivity.

Modal. $\mathcal{A}_{\text{modal}} = \{\texttt{text}, \texttt{image}\}$. Image tokens enter the language model from the vision encoder rather than the embedding table, so their KV can have different statistics, such as per-channel variance and sparsity, and can respond differently to low-bit quantization.

Semantic. $\mathcal{A}_{\text{semantic}} = \{\texttt{inst}, \texttt{user}, \texttt{assistant}, \texttt{reasoning}, \texttt{tool_call}, \texttt{obs}, \texttt{delim}\}$. These are functional roles in the agent loop, summarized in Table 1. Tags that never occur, such as $(\cdot, \texttt{image}, \texttt{reasoning})$, have zero token count and drop out of the optimization. All three axes are determined by the chat template alone, without inspecting token content or running the model, so the tagger runs cheaply during request preprocessing.

3.1.1 Allocation Setup

Consider a request whose prefill contains N tokens. Each token t contributes a key vector and a value vector to the KV cache, and receives a tag $s(t) \in \mathcal{S}$. We assign each tag k a bit-width $b_k \in \{2, 4\}$

shared by all tokens with that tag. Our goal is to minimize the perturbation of the attention output subject to a target average bit-width over the prefill tokens.

3.1.2 Allocation Objective

Let o_i denote the full-precision attention output at query position i, and $\tilde{o}_i(\mathbf{b})$ the output when KV vectors are quantized according to a per-tag bit-width allocation $\mathbf{b} = (b_k)_{k \in \mathcal{S}}$. We aim to find the allocation $\mathbf{b}$ that minimizes the expected MSE of attention output:

$$\mathcal{L}(\mathbf{b}) \triangleq \mathbb{E}_i \left[\|o_i - \tilde{o}_i(\mathbf{b})\|_2^2 \right]. \tag{2}$$

We target the attention output MSE rather than KV reconstruction MSE because the two are not monotonically related: a token with small KV quantization error can still contribute large output error if attention concentrates on it, whereas a large error on a rarely attended token has little effect. To make this MSE tractable, we expand the attention output to first order in the quantization noise δ, yielding an expression linear in the per-token noise. We assume δ is zero-mean and uncorrelated across tags. The cross-tag terms then vanish in expectation, and the loss decomposes over tags:

$$\hat{\mathcal{L}}(\mathbf{b}) = \sum_{k \in \mathcal{S}} D_k(b_k), \tag{3}$$

where $D_k(b)$ is the output MSE when only tokens with tag k are quantized to b bits and all other tokens remain full precision. Under this approximation, $D_k(b)$ depends only on the tag k and its assigned bit-width b. We measure $D_k(b)$ directly from calibration data: on a calibration capture of KV cache, we quantize only tokens with tag k at bit-width b, replay attention, and record the output MSE against the full-precision reference. Therefore, our analysis produces a table of $2 \cdot |\mathcal{S}|$ distortion values, one per tag and bit-width, which is reused across bit-width budgets during optimization.

3.1.3 Optimization

Let $N_k = |\{t : s(t) = k\}|$ denote the token count for tag k in the KV cache. We estimate N_k in two ways. For `inst` tags, which cover the system prompt and tool schemas, tokens are shared across most, if not all, requests under prefix caching, and the cache stores a single copy. Therefore, we take N_k as the median token count in `inst`-semantic segments across calibration requests. For all other tags, we sum the prefill token count across all calibration requests. With target average bit-width $B \in [2, 4]$, the allocation problem is:

$$\min_{\mathbf{b} \in \{2,4\}^{|\mathcal{S}|}} \sum_{k \in \mathcal{S}} D_k(b_k) \qquad \text{s.t.} \qquad \sum_{k \in \mathcal{S}} N_k \, b_k \;\le\; B \sum_{k \in \mathcal{S}} N_k. \tag{4}$$

Upgrading tag k from 2 to 4 bits costs $2N_k$ extra bits in the cache and reduces the total MSE by $D_k(2) - D_k(4)$. We define the per-bit gain of this upgrade as:

$$\rho_k \triangleq \frac{D_k(2) - D_k(4)}{2\, N_k}. \tag{5}$$

When at most 22 tags are active ($N_k \ge 1$), we enumerate all allocation configurations and select the one with the minimal total MSE. For larger numbers of active tags, we take a greedy approach: sort tags by ρ_k in descending order, walk the sorted list, and upgrade each tag whose extra cost $2N_k$ fits in the remaining budget. Algorithm 1 in Appendix A describes our greedy allocation.

3.1.4 Calibration

For each model and target workload, we take 5% of the dataset to form a calibration set, and use it to compute the per-tag bit-width allocation in three stages.

KV capture. We run the calibration set through the serving system to capture the KV traces. For each model, we select four to six layers, evenly spread across the model's depth, and register forward hooks on each layer's attention module, which receives Q, K, and V after RoPE [37]. The hooks fire on prefill only, capturing Q for the new tokens and KV of the prefill sequence.

Sensitivity analysis. For each capture and each tag k active in that capture, we quantize the KV of tokens tagged with k using bit-width $b \in \{2, 4\}$, compute the attention output, and record the MSE against the full-precision reference. This yields a raw $D_k^{\ell,r,h}(b)$ for every layer ℓ, request r, and head h. We aggregate these into a final sensitivity score $D_k(b)$ as follows:

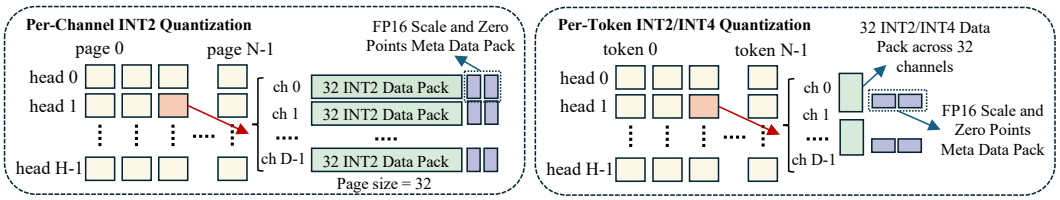


Figure 4: KV cache buffer layouts. Left: per-channel INT2 keys, indexed by (page, head). Right: per-token INT2 values and INT4 KV, indexed by (token, head). Page size matches the quantization group size G = 32. Quantized data are packed into unsigned INT8 buffers.

1. **Max across heads.** We take the maximum error across heads to ensure that no single attention subspace becomes a bottleneck, keeping the most sensitive head functional.
2. **Mean across requests.** We average the results over all calibration requests to ensure the metric is representative and stable across different inputs.
3. **Sum across layers.** We sum the errors across all layers to capture their cumulative impact, assuming that quantization noise is additive.

Bit-width allocation and budget selection. The aggregated sensitivities $D_k(b)$ and token counts N_k are passed to the optimization algorithm (Section 3.1.3) to produce the per-tag allocation $\mathbf{b}^\star$ for a target average bit-width B. We choose B by sweeping $B \in [2, 4]$ on the calibration set and selecting the smallest value at which accuracy remains comparable to the BF16 baseline. Since both $D_k(b)$ and N_k are pre-computed, re-running bit-width allocation is highly efficient.

3.2 System Design and Implementation

This section describes how TriAxialKV is implemented in a serving system: the quantization scheme and buffer layout, the mixed-precision INT2 and INT4 memory pools, and the request lifecycle, which applies the per-tag allocation from Section 3.1 and runs the prefill and fused decode kernels.

3.2.1 Quantization Scheme and Buffer Layout

Bit-widths and quantization axes. Figure 4 shows our quantization scheme and packed buffer layout. We use an asymmetric group-wise quantization scheme with group size $G = 32$ throughout. For INT4, we quantize *both keys and values per-token*. Channels in a head are split into groups of G elements, each quantized independently with its own (scale, zero-point) pair. For INT2, we *quantize keys per-channel and values per-token*. Keys in a quantization group share one (scale, zero-point) pair *for each channel*. We quantize INT2 keys per-channel because, at low bit-width, high-magnitude outliers in some channels inflate the scale and destroy resolution across other channels under per-token quantization [27]. Values are well-behaved under per-token quantization at both bit-widths.

Buffer layout. The INT2 key buffer, which uses per-channel quantization, is indexed by `(page, head)`. As shown in Figure 4, each entry holds the quantized keys for all G tokens of the page under that head, grouped by channel, along with scales and zero-points. This ensures that each channel's keys for the G tokens are contiguous in memory, enabling efficient coalesced loads. For per-token INT4 KV and INT2 values, each buffer is instead indexed by `(token, head)`. Each entry stores the quantized data for all channels under one `(token, head)`, followed by their scales and zero-points. Quantized data are stored as unsigned INT8 buffers with four INT2 or two INT4 values packed per byte, whereas scales and zero-points are stored in FP16.

Residual policy for INT2 keys. Per-channel quantization of INT2 keys requires the scale and zero-point of a channel to be shared across a group of G tokens. For the residual tokens that do not form a group, we *route them through the INT4 per-token path* instead. In KIVI [27], the residual tokens are stored in full precision in a dedicated buffer, but we do not take this approach for two reasons. First, adding a residual buffer complicates memory allocation and forces the attention kernel to handle three precisions. Second, KIVI's residual buffer holds up to 128 tokens' KV in full precision per request in every layer. This adds considerable overhead and erodes the memory savings that our

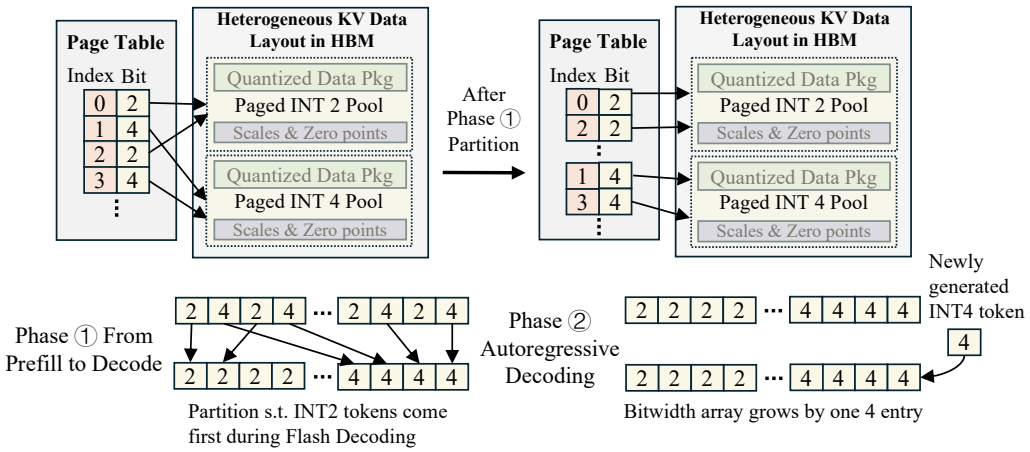


Figure 5: Per-request page table management. Phase 1 partitions page table entries so INT2 pointers precede INT4, enabling bit-width-homogeneous flash-decoding splits; Phase 2 generates INT4 tokens autoregressively during decode and grows the page table by one entry per step.

INT2 path is meant to deliver, especially when there are many short requests. Keeping the residual tokens on the INT4 path makes memory management simple and reduces overhead.

3.2.2 Mixed-Precision Memory Pool

Shared address space. Both the INT2 and INT4 memory pools *share a single virtual address space*, split by an offset. The offset is set at startup from the calibrated average bit-width B, which sets the ratio of INT2 and INT4 tokens. Addresses below the offset index the INT2 pool, and addresses at or above the offset index the INT4 pool. Downstream components, such as the attention backend, can infer a token's precision by comparing its memory address against the offset.

Memory allocation. Each pool has its own page allocator for memory allocation and freeing. When a request enters the scheduler, the bit-width tagger produces a per-token bit-width array. INT2 and INT4 tokens are routed to their respective allocators for memory allocation, and each allocator returns free page indices from its own list. No cross-pool coordination is required.

3.2.3 Request Lifecycle

Tagging. At serving time, each incoming request is tagged on the CPU during request scheduling and *assigned an array of per-token bit-widths* that propagates to the downstream memory allocator and attention backend. Specifically, we run the triaxial tagger from Section 3.1 on the tokenized input to produce a per-token tag code. Each tag code is looked up in the bit-width allocation map $\mathbf{b}^\star : \mathcal{S} \to \{2, 4\}$ from the calibration, yielding a per-token bit-width array attached to the request. To keep the tagger general across models and workloads, we drive it entirely by chat-template structure. We use special-token IDs from the tokenizer, such as role markers, message delimiters, and structural brackets for thinking and tool calls, to segment the sequence with a single scan.

Prefill. For prefill, we implement a Triton [39] gather-dequant kernel that reads the matched-prefix slot indices from the INT2 and INT4 pools, unpacks each slot's payload, dequantizes it into the activation precision (BF16), and writes the results into a buffer. The buffer is then passed to the FlashInfer [49] prefill attention kernel without further modification. Queries, keys, and values for the newly computed tokens are passed directly to the prefill kernel alongside the dequantized prefix.

Decode. After prefill, the decode stage generates new tokens. As shown in Figure 5, each request's KV state is tracked in a per-request page table that records per-token pointers to the memory pool. In phase 1, we *partition* the pointers so that all INT2 pointers precede the INT4 pointers. This suits our custom fused decode kernel, which follows Flash-Decoding [13]: the KV is processed in parallel contiguous splits whose partial outputs are merged with a log-sum-exp reduction. Our partition

Table 2: Task accuracy (%) on BFCL Memory, reported as mean and standard deviation over three runs. TriAxialKV Mixed stores 75% of prefill tokens at INT2 (70% for Qwen3-14B).

Method	Falcon3-10B-Instruct	Qwen3-14B	Qwen3-32B	Qwen3-235B-A22B-FP8
SGLang BF16 [55]	25.04±1.26	25.41±1.36	25.19±1.22	23.78±0.89
SGLang FP4 [55]	26.37±2.65	21.19±2.76	20.52±0.34	20.96±0.46
KIVI [27]	22.52±0.46	19.41±1.99	20.74±1.23	20.37±2.06
OSCAR [57]	21.33±1.68	9.34±1.60†	24.30±0.71	18.52±3.04
TriAxialKV INT4	24.30±2.34	22.15±2.38	23.93±0.93	24.44±1.18
TriAxialKV Mixed	23.04±2.27	23.48±2.45	24.00±1.11	22.59±1.28

†Affected by a head-indexing bug in OSCAR's decode kernel that is triggered by Qwen3-14B's attention-head configuration.

ensures that the leading splits handle INT2 KV and the trailing ones handle INT4. Each split is bit-width-homogeneous and runs a single dequantization path. In phase 2, each autoregressive decode step generates a new token, whose KV is stored in INT4. Our fused dequantization-attention kernel has two paths: INT2 and INT4. Each path loads only the required KV data, unpacks it via bit shifts, and applies dequantization on the fly during attention computation. Keys and values of the generated token are written to the INT4 pool, and its address is appended to the page table.

4 Experimental Results

4.1 Experimental Setup

Benchmarks and models. We evaluate TriAxialKV on the memory category of the Berkeley Function Calling Leaderboard (BFCL) [31] and OSWorld [46], which cover two representative agentic regimes: text-only function calling with structured tool schemas, and multimodal computer-use trajectories with screenshot observations. On BFCL, we evaluate Qwen3 [47] (14B, 32B, and 235B-A22B-Instruct-2507-FP8) and Falcon3-10B-Instruct [15]. On OSWorld, we evaluate Qwen3-VL [5] (8B-Thinking, 32B-Thinking) and InternVL3.5-38B [42].[2] For the axis ablation, we additionally evaluate Qwen3.5-4B [33] on Claw-Eval [48], a multimodal agentic benchmark.

Baselines. We compare against four baselines covering the relevant prior art. SGLang [55] BF16 serves the model with full-precision KV cache and is the lossless reference. SGLang FP4 uses SGLang's built-in FP4 KV cache backend, representing uniform low-bit floating-point quantization. We further compare with two 2-bit KV cache quantization methods: KIVI [27], a seminal asymmetric scheme, and OSCAR [57], a recent rotation-based scheme. We also report TriAxialKV INT4, which stores all tokens at INT4, to isolate the effect of the per-tag INT2/INT4 allocation.

End-to-end serving performance. We measure end-to-end serving throughput in tokens per second (TPS) on Qwen3-VL-8B and Qwen3-VL-32B, running OSWorld trajectories. The throughput is measured on both a single NVIDIA B200 GPU (180 GB HBM3e) and a single NVIDIA H100 GPU (80 GB HBM3), both hosted in our internal cluster. For each (model, GPU) configuration, we measure the throughput at the largest possible batch size without running out of memory.

4.2 Task Accuracy

Tables 2 and 3 report task accuracy on BFCL Memory and OSWorld. Across all experiments, our mixed-precision allocation *tracks the BF16 reference closely*. On BFCL Memory, with 75% of prefill tokens at INT2 (70% on Qwen3-14B), TriAxialKV stays within 2.0% of the BF16 baseline on every model. On OSWorld, with 85% of prefill tokens at INT2, TriAxialKV matches the BF16 accuracy on every model. TriAxialKV Mixed also *performs on par with INT4*, showing that moving most tokens to INT2 under the calibrated allocation costs little accuracy. Appendix C gives per-category accuracy and OSWorld confidence intervals, and Appendix D gives output lengths.

The quantization baselines are less consistent across models. On BFCL Memory, SGLang FP4 matches BF16 on Falcon3-10B but drops 4–5 points on Qwen3-14B and Qwen3-32B. KIVI falls

[2]The licenses for all models, benchmarks, and libraries used in this work are listed in Appendix H.

Table 3: Task accuracy (%) on OSWorld across multimodal models. Gray numbers in brackets indicate deltas relative to the BF16 baseline. TriAxialKV Mixed maintains the accuracy of the BF16 reference, with a substantially lower memory budget.

Method	Qwen3-VL-8B-Thinking	Qwen3-VL-32B-Thinking	InternVL3.5-38B
SGLang BF16 [55]	35.70 (+0.00)	39.69 (+0.00)	12.22 (+0.00)
SGLang FP4 [55]	29.52 (-6.18)	40.54 (+0.85)	12.13 (-0.09)
KIVI [27]	34.61 (-1.09)	41.49 (+1.80)	12.27 (+0.05)
OSCAR [57]	30.52 (-5.18)	40.20 (+0.51)	11.95 (-0.27)
TriAxialKV INT4	35.38 (-0.32)	41.43 (+1.74)	13.11 (+0.89)
TriAxialKV Mixed	35.78 (+0.08)	40.59 (+0.90)	12.77 (+0.55)

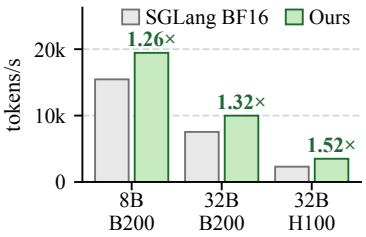


Figure 6: End-to-end throughput on OSWorld trajectories. TriAxialKV delivers 1.26–1.52× the throughput of SGLang BF16.

Table 4: End-to-end serving comparison on Qwen3-VL-32B-Thinking, measured on a single NVIDIA B200 GPU by replaying OSWorld trajectories. A more comprehensive comparison, including bytes/token, residual windows, and absolute capacity, is provided in Appendix E.

Method	Prefill	Decode	Bits/ elem.	Capacity vs. BF16	Throughput (tokens/s)	Speedup
SGLang BF16	FlashInfer	SGLang Triton	16.0	1.00×	7,787	1.00×
SGLang BF16	SGLang Triton	SGLang Triton	16.0	1.00×	5,983	0.77×
SGLang FP4	TRT-LLM	TRT-LLM	4.5	3.16×	4,287	0.55×
KIVI†	SGLang Triton	SGLang Triton	3.0	1.00×	5,772	0.74×
OSCAR	FA-4	OSCAR Triton	2.5	5.70×	7,630	0.98×
TriAxialKV INT4	FlashInfer	Ours Triton	5.0	3.18×	9,407	1.21×
TriAxialKV Mixed	FlashInfer	Ours Triton	3.3	4.51×	**10,274**	**1.32**×

†KIVI has no official serving-system integration. We reimplemented it on SGLang with fake quantization to measure its task accuracy, so its compressed footprint yields neither capacity nor bandwidth gains.

below BF16 on every model by 2.5%–6.0%, and OSCAR has a competitive accuracy only on Qwen3-32B. On OSWorld, Qwen3-VL-8B is notably more sensitive to KV quantization: SGLang FP4 and OSCAR degrade accuracy by 6.2% and 5.2%, and KIVI by 1.1%. Meanwhile, TriAxialKV stays close to BF16 on every model across both benchmarks.

Comparing TriAxialKV Mixed against the two 2-bit baselines, KIVI and OSCAR, *demonstrates the value of per-tag allocation*. Both baselines apply 2-bit quantization to all non-residual tokens, including the system prompt and tool schemas, which our calibrated allocation keeps at INT4. These tokens carry the function signatures and argument specifications that the model *must reproduce verbatim in its tool calls*, so quantization errors in their KV can plausibly propagate into incorrect argument names or types, which BFCL Memory scores as task failures. TriAxialKV Mixed spends the 4-bit budget on exactly this `inst`-tagged segment while pushing less sensitive segments to 2-bit, recovering most of the accuracy that uniform 2-bit quantization gives up.

4.3 Serving Throughput

To measure the serving throughput, we replay OSWorld trajectories, which have an average prefill length of 11,000 tokens and an average decode length of 300 tokens. We compare TriAxialKV with the SGLang BF16 baseline. Both methods use FlashInfer [49] for prefill; the SGLang BF16 baseline uses SGLang's original Triton decode kernel, while our method uses our custom fused Triton kernel for mixed-precision dequantization and attention. Figure 6 reports end-to-end throughput at the maximum concurrency each method can sustain. On a single B200 GPU, TriAxialKV delivers *1.32× the BF16 throughput* on Qwen3-VL-32B, and *1.26*× on Qwen3-VL-8B. On a single H100 GPU, the gain on Qwen3-VL-32B rises to *1.52*×. It also supports 3.4–4.0× as many concurrent requests.

We also compared TriAxialKV with other baselines on a single B200 GPU. As shown in Table 4, TriAxialKV INT4 and SGLang FP4 have a similar capacity, but TriAxialKV INT4 delivers 2.2× the throughput of the TRT-LLM FP4 backend in SGLang. Moving from INT4 to mixed allocation, TriAxialKV has a further 1.4× increase in capacity and increases the speedup to 1.32×. OSCAR

Table 5: Drop-one-axis ablation. Removing any applicable axis degrades task performance.

	BFCL Memory		Claw-Eval
Variant	Qwen3-14B	Qwen3-32B	Qwen3.5-4B
No Temporal	22.00	24.00	31.60
No Modal	–	–	32.18
No Semantic	18.00	20.89	33.10
Full	**24.22**	**25.11**	**34.17**

Table 6: Memory budget sweep for Qwen3-14B on BFCL Memory.

INT2 Tokens	Accuracy (%)	Δ vs. BF16
100%	19.56	−5.85
90%	20.37	−5.04
80%	20.08	−5.33
75%	21.63	−3.78
70%	**23.48**	−1.93
BF16	25.41	0.00

holds the largest KV cache capacity but shows no end-to-end speedup. In OSCAR's implementation, each prefill rotates the QKV, dequantizes the cached prefix to BF16, and assembles each request's KV in a Python loop that forces GPU–CPU synchronization in every layer. We attribute the missing speedup to this overhead, which offsets the capacity gain on this prefill-heavy workload.

4.4 Ablation Study

Axis ablation. Table 5 removes one axis of the taxonomy at a time, on BFCL Memory and Claw-Eval. Dropping an axis merges all tags that differ only along that axis into one, and we re-run calibration and allocation on the reduced tag space at the same average bit-width. *No Temporal* treats all tokens as belonging to a single turn, removing temporal recency from the tag space. *No Modal* treats image and text tokens as the same. *No Semantic* treats all tokens as a single semantic role, regardless of whether they are system prompts, user queries, or tool calls.

Removing any axis lowers accuracy, but which axis matters most depends on the workload. On BFCL Memory, the semantic axis dominates: removing it leads to a degradation of 6.22% on Qwen3-14B and 4.22% on Qwen3-32B, three to four times the cost of removing the temporal axis. The semantic axis gives the system prompt and tool schemas their own tags, so the allocator can keep them at INT4. Errors in these segments propagate into incorrect function and argument names, which cause the task to fail. Without the semantic axis, these tokens are merged with other roles that share their temporal and modality tags, and the merged tag is assigned INT2. On Claw-Eval, the order reverses: removing temporal recency costs the most, followed by modality and semantic role. The dominant axis shifts with the workload, which is why the taxonomy keeps all three rather than specializing to one.

Memory budget. Table 6 sweeps the share of prefill tokens stored at INT2 on Qwen3-14B (BFCL Memory), from uniform INT2 to 70%. Moving the first 20% of tokens to INT4 leaves it essentially unchanged, while the next 10% recovers most of the gap, reaching 23.48%. Because the budget at which accuracy recovers differs across models and workloads, a single fixed budget either wastes memory or falls short of it, which is why we calibrate it per model and workload.

5 Conclusion

We presented TriAxialKV, a mixed-precision KV-cache quantization framework that exploits the structure of agentic prefills along three complementary axes: temporal recency, modality, and semantic role. TriAxialKV tags each prefill token using only the chat template, calibrates the attention-output distortion of each tag, and allocates INT2/INT4 bit-widths under a memory budget. It runs in SGLang with custom paged mixed-precision memory pools and fused Triton decode kernels. On BFCL Memory and OSWorld, TriAxialKV preserves the task accuracy relative to the BF16 reference. On Qwen3-VL-32B (OSWorld), it provides 4.5× KV-cache capacity and delivers 1.32× the BF16 throughput on B200 and 1.52× on H100. These results show that jointly reasoning about temporal, modal, and semantic structure is both necessary and practical for aggressive KV-cache compression.

Acknowledgments and Disclosure of Funding

We thank Taowen Liu and Can Xiao for their helpful discussions. This work was supported by the Advanced Research and Invention Agency (ARIA) under the Scaling Compute Program.

References

[1] Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the efficiency of computer-use agents. In *ICML Workshop on Computer Use Agents*, 2025.

[2] Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. In *COLM*, 2025.

[3] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *OSDI*, 2024.

[4] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated LLMs. In *NeurIPS*, 2024.

[5] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[6] Debarag Banerjee, Pooja Singh, Arjun Avadhanam, and Saksham Srivastava. Benchmarking llm-powered chatbots: Methods and metrics. *arXiv preprint arXiv:2308.04624*, 2023.

[7] Sayed Pedram Haeri Boroujeni, Niloufar Mehrabi, Patrick Woods, Gabriel Hillesheim, and Abolfazl Razi. Don't waste bits! adaptive KV-cache quantization for lightweight on-device llms. *arXiv preprint arXiv:2604.04722*, 2026.

[8] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. In *COLM*, 2025.

[9] Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *ICLR*, 2025.

[10] 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.

[11] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

[12] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022.

[13] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for long-context inference. https://crfm.stanford.edu/2023/10/12/flashdecoding.html, 2023. Stanford CRFM Blog.

[14] Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Christos Kozyrakis. Liminal: Exploring the frontiers of llm decode performance. *arXiv preprint arXiv:2507.14397*, 2025.

[15] Falcon-LLM Team. The falcon 3 family of open models. https://huggingface.co/blog/falcon3, 2024.

[16] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. In *NeurIPS*, 2025.

[17] 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. In *ICLR*, 2024.

[18] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *ACL*, 2024.

[19] Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. In *NeurIPS*, 2024.

[20] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *NeurIPS*, 2024.

[21] Jinda Jia, Jisen Li, Zhongzhu Zhou, Jung Hwan Heo, Jue Wang, Tri Dao, Shuaiwen Leon Song, Ben Athiwaratkun, Chenfeng Xu, Tianyi Zhang, and Xiaoxia Wu. Saw-int4: System-aware 4-bit kv-cache quantization for real-world llm serving. *arXiv preprint arXiv:2604.19157*, 2026.

[22] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

[23] 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. In *NeurIPS*, 2024.

[24] Chengxi Liao and Zeyi Wen. Channel-aware mixed-precision quantization for efficient long-context inference. In *ICLR*, 2026.

[25] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. In *MLSys*, 2025.

[26] Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. PM-KVQ: Progressive mixed-precision KV cache quantization for long-cot LLMs. In *ICLR*, 2026.

[27] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *ICML*, 2024.

[28] Jiayi Nie, Haoran Wu, Yao Lai, Zeyu Cao, Cheng Zhang, Binglei Lou, Erwei Wang, Jianyi Cheng, Timothy M. Jones, Robert Mullins, Rika Antonova, and Yiren Zhao. Kernelcraft: Benchmarking for agentic close-to-metal kernel generation on emerging hardware. *arXiv preprint arXiv:2603.08721*, 2026.

[29] NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2023.

[30] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *ISCA*, 2024.

[31] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *ICML*, 2025.

[32] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *FAST*, 2025.

[33] Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026.

[34] Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. In *ICLR*, 2026.

[35] Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. Longcodebench: Evaluating coding LLMs at 1m context windows. In *COLM*, 2025.

[36] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *NeurIPS*, 2024.

[37] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

[38] Qian Tao, Wenyuan Yu, and Jingren Zhou. AsymKV: Enabling 1-bit quantization of KV cache with layer-wise asymmetric quantization configurations. In *COLING*, 2025.

[39] Philippe Tillet and David Cox. 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*, 2019.

[40] Dezhan Tu, Danylo Vashchilenko, Yuzhe Lu, and Panpan Xu. VL-cache: Sparsity and modality-aware KV cache compression for vision-language model inference acceleration. In *ICLR*, 2025.

[41] Zhongwei Wan, Ziang Wu, Che Liu, Jinfa Huang, Zhihong Zhu, Peng Jin, Longyue Wang, and Li Yuan. LOOK-M: Look-once optimization in KV cache for efficient multimodal long-context inference. In *Findings of EMNLP*, 2024.

[42] Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025.

[43] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *KDD*, 2025.

[44] Haoran Wu, Can Xiao, Jiayi Nie, Xuan Guo, Binglei Lou, Jeffrey T.H. Wong, Zhiwen Mo, Cheng Zhang, Przemyslaw Forys, Chengyang Ai, Timi Adeniran, Wayne Luk, Hongxiang Fan, Jianyi Cheng, Timothy M. Jones, Rika Antonova, Robert Mullins, and Aaron Zhao. Combating the memory walls: Optimization pathways for long-context agentic llm inference. In *ISCA*, 2026.

[45] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.

[46] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *NeurIPS*, 2024.

[47] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[48] Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Towards trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

[49] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. In *MLSys*, 2025.

[50] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *OSDI*, 2022.

[51] Ted Zadouri, Markus Hoehnerbach, Jay Shah, Vijay Thakkar, and Tri Dao. Flashattention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling. In *MLSys*, 2026.

[52] 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. In *NeurIPS*, 2023.

[53] Yi Zhao, Yajuan Peng, Nguyen Cam-Tu, Zuchao Li, Xiaoliang Wang, Hai Zhao, and Xiaoming Fu. Smallkv: Small model assisted compensation of kv cache compression for efficient llm inference. In *NeurIPS*, 2025.

[54] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In *ICLR*, 2024.

[55] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. In *NeurIPS*, 2024.

[56] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *OSDI*, 2024.

[57] Zhongzhu Zhou, Donglin Zhuang, Jisen Li, Ziyan Chen, Shuaiwen Leon Song, Ben Athiwaratkun, and Xiaoxia Wu. Oscar: Offline spectral covariance-aware rotation for 2-bit kv cache quantization. *arXiv preprint arXiv:2605.17757*, 2026.

A Greedy Bit-Width Allocation

Algorithm 1 Triaxial bit-width allocation

Require: Tag set $\mathcal{S}$; counts $\{N_s\}$; distortions $\{D_s(2), D_s(4)\}$; budget B.
Ensure: Allocation $\mathbf{b}^\star \in \{2,4\}^{|\mathcal{S}|}$.

```
Compute ρ_s ← (D_s(2) − D_s(4)) / (2N_s) for all s ∈ S.
Sort S by ρ_s descending; set b_s ← 2 and R ← (B − 2) Σ_s N_s.
for s in sorted order do
    if 2N_s ≤ R then
        b_s ← 4;  R ← R − 2N_s.
    end if
end for
return b* = (b_s)_{s∈S}.
```

B Baseline Selection

Agentic benchmarks run multi-turn episodes against a live serving engine, so a KV-cache method needs a serving integration to be evaluated on them, and efficient kernels for a throughput comparison to be meaningful. Table 7 lists the candidates we considered. We select SGLang FP4, the built-in low-bit KV cache of the framework we build on; OSCAR, the only 2-bit method in the table that is both open source and integrated with a serving system; and KIVI, a widely used asymmetric 2-bit scheme, which we reimplement in SGLang with fake quantization to measure its accuracy. KVQuant, PM-KVQ, and LOOK-M release code without a serving integration, and ThinKV and VL-Cache have no public code.

Table 7: KV-cache compression methods considered as baselines. *Serving integration*: runs inside a serving engine, which agentic benchmarks require for end-to-end evaluation. *End-to-end speedup*: the original work reports serving-throughput gains. *Residual buffer*: keeps a full-precision window of tokens per request, whose memory does not shrink with the bit-width.

Method	Open source	Serving integration	End-to-end speedup	Residual buffer
SGLang FP4 [55]	✓	✓	✗	✗
KIVI [27]	✓	✗	✗	✓
KVQuant [20]	✓	✗	✗	✗
QServe [25]	✓	✓	✓	✗
PM-KVQ [26]	✓	✗	✗	✗
ThinKV [34]	✗	✓	✓	✓
VL-Cache [40]	✗	✓	✓	✗
LOOK-M [41]	✓	✗	✗	✗
OSCAR [57]	✓	✓	✓	✓
TriAxialKV (Ours)	✓	✓	✓	✗

C Per-Category Accuracy on BFCL and OSWorld

BFCL Memory evaluates whether an agent can store and recall information across sessions with one of three memory backends: a key-value store (KV), a vector store (Vector), and recursive summarization (RecSum). OSWorld groups its tasks by application domain: OS (operating-system and terminal tasks), Office (LibreOffice Calc, Writer, and Impress), Daily (Chrome, VLC, and Thunderbird), Professional (VS Code and GIMP), and Workflow (tasks that span several applications). Several categories are small; OS, for example, has about two dozen tasks, so a single task moves its accuracy by more than 4 points. Table 10 therefore also reports 95% bootstrap confidence intervals for overall OSWorld accuracy.

Table 8: Per-category accuracy (%) on BFCL Memory across the KV, Vector, and RecSum sub-tasks, averaged over the same three runs as Table 2; Overall is the mean of the three sub-tasks.

Method	Overall	KV	Vector	RecSum
Falcon3-10B-Instruct				
SGLang BF16 [55]	25.04	10.45	16.22	48.44
SGLang FP4 [55]	26.37	12.89	16.45	49.78
KIVI [27]	22.52	11.78	18.00	37.78
OSCAR [57]	21.33	11.33	17.33	35.33
TriAxialKV INT4	24.30	10.89	16.45	45.56
TriAxialKV Mixed	23.04	12.22	15.11	41.78
Qwen3-14B				
SGLang BF16 [55]	25.41	12.00	18.89	45.33
SGLang FP4 [55]	21.19	9.33	11.78	42.44
KIVI [27]	19.41	5.33	14.00	38.89
OSCAR [57]	9.34	8.44	8.22	11.33
TriAxialKV INT4	22.15	8.44	14.67	43.33
TriAxialKV Mixed	23.48	11.78	13.78	44.89
Qwen3-32B				
SGLang BF16 [55]	25.19	16.22	15.11	44.22
SGLang FP4 [55]	20.52	11.11	16.00	34.44
KIVI [27]	20.74	12.67	15.78	33.78
OSCAR [57]	24.30	13.78	17.56	41.56
TriAxialKV INT4	23.93	14.67	16.44	40.66
TriAxialKV Mixed	24.00	11.11	20.22	40.67
Qwen3-235B-A22B-Instruct-2507-FP8				
SGLang BF16 [55]	23.78	9.56	16.67	45.11
SGLang FP4 [55]	20.96	7.56	11.11	44.22
KIVI [27]	20.37	11.78	12.89	36.44
OSCAR [57]	18.52	10.22	14.89	30.44
TriAxialKV INT4	24.44	13.33	14.22	45.78
TriAxialKV Mixed	22.59	10.00	14.45	43.33

Table 9: Per-category accuracy (%) on OSWorld by application category.

Method	Overall	OS	Office	Daily	Professional	Workflow
Qwen3-VL-8B-Thinking						
SGLang BF16 [55]	35.70	47.83	29.67	51.21	48.94	19.65
SGLang FP4 [55]	29.52	50.00	25.77	43.08	31.82	13.06
KIVI [27]	34.61	43.48	30.30	44.27	51.06	20.86
OSCAR [57]	30.52	34.78	26.94	46.90	42.55	13.73
TriAxialKV INT4	35.38	39.13	33.28	50.41	48.94	17.34
TriAxialKV Mixed	35.78	43.48	32.97	52.72	40.43	21.00
Qwen3-VL-32B-Thinking						
SGLang BF16 [55]	39.69	52.17	33.58	53.73	48.94	27.30
SGLang FP4 [55]	40.54	43.48	39.22	56.43	48.94	23.60
KIVI [27]	41.49	56.52	39.21	47.10	57.45	27.21
OSCAR [57]	40.20	54.55	38.34	48.92	48.94	26.68
TriAxialKV INT4	41.43	52.17	44.59	49.51	48.94	23.94
TriAxialKV Mixed	40.59	43.48	37.41	52.59	53.19	27.35
InternVL3.5-38B						
SGLang BF16 [55]	12.22	21.74	6.30	21.51	21.28	4.55
SGLang FP4 [55]	12.13	0.00	7.97	23.59	17.02	2.04
KIVI [27]	12.27	21.74	5.48	28.11	10.87	5.68
OSCAR [57]	11.95	17.39	9.93	21.57	12.77	4.55
TriAxialKV INT4	13.11	23.81	5.40	24.22	21.28	6.10
TriAxialKV Mixed	12.77	26.09	6.27	24.20	19.15	4.55

Table 10: OSWorld task accuracy (%) with 95% bootstrap confidence intervals in gray, obtained by resampling task-level results (10,000 resamples, percentile intervals).

Method	Qwen3-VL-8B-Thinking	Qwen3-VL-32B-Thinking	InternVL3.5-38B
SGLang BF16	35.70 (30.67–40.85)	39.69 (34.44–44.90)	12.22 (8.75–15.74)
SGLang FP4	29.52 (24.58–34.74)	40.54 (35.37–45.69)	12.13 (8.49–16.03)
KIVI	34.61 (29.72–39.68)	41.49 (36.32–46.74)	12.27 (8.81–15.84)
OSCAR	30.52 (25.78–35.30)	40.20 (35.20–45.31)	11.95 (8.73–15.44)
TriAxialKV INT4	35.38 (30.30–40.55)	41.43 (36.30–46.55)	13.11 (9.55–16.69)
TriAxialKV Mixed	35.78 (30.70–40.99)	40.59 (35.39–45.72)	12.77 (9.30–16.50)

D Output Length

KV-cache quantization can change generation behavior even when accuracy is preserved, for example by causing repetition or cutting reasoning short. Table 11 reports the mean number of generated tokens per task on both benchmarks. TriAxialKV Mixed stays within 3% of the BF16 output length on the three Qwen3 models of BFCL Memory, and within 7% on all OSWorld models.

Table 11: Mean generated tokens per task. (a) BFCL Memory, mean ± standard deviation over three runs. (b) OSWorld.

(a) BFCL Memory.

Method	Falcon3-10B	Qwen3-14B	Qwen3-32B	Qwen3-235B
SGLang BF16	546±35	1601±70	2317±52	715±26
SGLang FP4	657±40	1538±4	2353±20	822±31
KIVI	508±148	1690±29	2228±65	746±89
OSCAR	522±47	2189±131	2406±488	496±28
TriAxialKV INT4	463±81	1612±39	2197±5	697±11
TriAxialKV Mixed	455±25	1636±41	2312±106	695±38

(b) OSWorld.

Method	Qwen3-VL-8B	Qwen3-VL-32B	InternVL3.5-38B
SGLang BF16	10967	9351	3890
SGLang FP4	11671	10357	3460
KIVI	11063	10206	4017
OSCAR	10533	9279	4851
TriAxialKV INT4	9892	9542	3601
TriAxialKV Mixed	10196	9525	3874

E Full Serving Comparison

Table 12 extends Table 4 with per-token footprint, residual windows, and absolute capacity. Bits per element include quantization metadata: with group size $G = 32$ and an FP16 scale and zero-point per group, INT2 and INT4 cost 3 and 5 bits per element, so the 3.3 bits of TriAxialKV Mixed correspond to 85% of tokens at INT2. Bytes per token count keys and values across all layers and KV heads of Qwen3-VL-32B. The residual column gives the full-precision tokens each method keeps per request: KIVI keeps a 128-token recent window and OSCAR keeps 256 sink and recent tokens, whereas TriAxialKV routes residual tokens through the INT4 path (Section 3.2.1) and keeps none.

Table 12: Serving comparison of all KV cache methods on Qwen3-VL-32B-Thinking (single B200, replayed OSWorld traces). Bits/element and bytes/token include quantization metadata and exclude the full-precision residual window, listed separately in tokens per request. Capacity is the number of tokens the KV cache pool holds.

Method	Prefill backend	Decode backend	Bits/ element	Bytes/ token	Residual (tokens/req)	Capacity (tokens)	Capacity vs. BF16	Throughput (tokens/s)	Speedup
SGLang BF16	FlashInfer	SGLang Triton	16.0	262,144	–	325,952	1.00×	7,786.74	1.00×
SGLang BF16	SGLang Triton	SGLang Triton	16.0	262,144	–	326,003	1.00×	5,982.67	0.77×
SGLang FP4	TRT-LLM	TRT-LLM	4.5	73,728	0	1,030,720	3.16×	4,287.16	0.55×
KIVI†	SGLang Triton	SGLang Triton	3.0	49,152	128	326,003	1.00×	5,771.76	0.74×
OSCAR	FlashAttention-4	OSCAR Triton	2.5	40,960	256	1,856,808	5.70×	7,630.07	0.98×
TriAxialKV INT4	FlashInfer	Ours Triton	5.0	81,920	0	1,037,063	3.18×	9,407.18	1.21×
TriAxialKV Mixed	FlashInfer	Ours Triton	3.3	54,067	0	1,469,312	4.51×	**10,274.45**	**1.32**×

†KIVI has no official serving-system integration. We reimplemented it on SGLang with fake quantization to measure its task accuracy, so its compressed footprint yields neither capacity nor bandwidth gains.

F Limitations

Our method has a few limitations:

- The per-tag sensitivity table is calibrated per workload and per model, requiring a one-time offline pass over a small calibration set. While inexpensive, this means the method is not zero-shot transferable to entirely new workloads.
- Our tagger relies on chat-template structure such as role markers, message delimiters, and brackets, and requires adaptation for models or prompting styles that lack standardized markers.
- Our allocator currently selects between INT2 and INT4 only. Supporting finer-grained bit-widths could yield further Pareto improvements, but would complicate kernel and memory-pool design.

G Broader Impacts

TriAxialKV is a KV-cache compression technique that reduces the memory and energy cost of serving large language models. We see two primary **positive** societal impacts. First, lower memory consumption per request enables larger batch sizes and higher throughput on the same hardware, reducing the per-token energy cost of inference and the corresponding environmental footprint of large-scale LLM deployment. Second, by allowing capable models to fit within the memory budget of less-expensive GPUs, our method lowers the hardware barrier to deploying agentic systems, broadening access for researchers and practitioners with limited compute resources. As far as we are aware, our work does not introduce new negative societal impacts beyond those already inherent to the underlying models being served. TriAxialKV does not modify model weights, training data, or output behavior; it is a serving-system optimization that preserves task accuracy relative to the BF16 reference. Any risks associated with the deployment of LLMs or agentic systems are inherited from the base models, and existing safeguards for those models continue to apply.

H Licenses for Existing Assets

We list below the existing assets used in this work, along with their licenses. All assets are used in accordance with their respective license terms.

Models:

- Qwen3 (14B, 32B, 235B-A22B-Instruct-2507-FP8) [47]: Apache License 2.0.
- Qwen3-VL (8B-Thinking, 32B-Thinking) [5]: Apache License 2.0.
- Qwen3.5-4B [33]: Apache License 2.0.
- Falcon3-10B-Instruct [15]: TII Falcon-LLM License 2.0.
- InternVL3.5-38B [42]: Apache License 2.0.

Benchmarks:

- Berkeley Function Calling Leaderboard (BFCL) [31]: Apache License 2.0.

- OSWorld [46]: Apache License 2.0.
- Claw-Eval [48]: MIT License.

Serving frameworks and libraries:

- SGLang [55] (v0.5.15.post1): Apache License 2.0. Our implementation is built as a fork.
- FlashInfer [49]: Apache License 2.0. Used as the BF16 prefill attention backend.
- Triton [39]: MIT License. Used to implement our fused mixed-precision decode kernels.
- TensorRT-LLM [29]: Apache License 2.0. Provides the FP4 backend of the SGLang FP4 baseline.
- FlashAttention-4 [51]: BSD 3-Clause License. Prefill backend of the OSCAR baseline.
- KIVI [27]: MIT License. Reimplemented in SGLang as a baseline for comparison.
- OSCAR [57]: MIT License. Used as a baseline for comparison.

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: Our contributions and scope are summarized in the abstract and introduction. Both of them claim (1) identification of three complementary axes of KV-cache quantization sensitivity, (2) a taxonomy-driven calibration and allocation framework, and (3) end-to-end system integration with measured throughput and accuracy gains.

   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: We discuss limitations in Appendix F, covering per-workload calibration overhead, the tagger's reliance on chat-template structure, and the restriction of the allocator to INT2/INT4 bit-widths.

   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: [Yes]

Justification: the main technical components are our sensitivity analysis and bit-width allocation. The assumptions and proofs are given in Section 3.1.

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: Section 3 fully describes the method, and Section 4.1 specifies the datasets, models, and hardware needed to reproduce the 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: code will be released shortly after paper publication. The method is described in sufficient detail to enable independent reimplementation, and all datasets and base models used are publicly available.

   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: Section 4.1 specifies models, datasets, baselines and hardware. Section 4.3 describes the throughput measurement protocol. Calibration details are described in Section 3.1.4.

   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: On BFCL Memory, Table 2 reports the mean and standard deviation over three independent runs of every configuration, and Table 8 averages its per-category results over the same runs. OSWorld trajectories are too expensive to repeat for every configuration, so Table 10 reports 95% percentile bootstrap confidence intervals obtained by resampling task-level outcomes 10,000 times.

   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 compute resources, including GPU specifications, are described in § 4.1.

   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 with the NeurIPS Code of Ethics. The work is on inference efficiency for existing publicly released models and standard public benchmarks, and does not involve human subjects, sensitive data collection, or deployment decisions affecting individuals.

   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: We discuss broader societal impacts in Appendix G. The primary positive impacts are reduced energy cost of LLM inference and lower hardware barriers to deploying capable models. We are not aware of negative societal impacts introduced by our compression technique beyond those already inherent to the underlying models being served.

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 new pre-trained models, generative data, or datasets.

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: All models, benchmarks, and software libraries used are publicly available and properly cited. A complete list of assets and their licenses is provided in Appendix H.

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 implementation will be released after paper acceptance. We introduce a new system implementation: an SGLang fork containing the triaxial tagger, calibration scripts, mixed-precision INT2/INT4 memory pool, and fused Triton decode kernels. Documentation will be included in the release.

    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 or research with human subjects. All evaluations are on existing public benchmarks.

    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 human subjects research.

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: LLMs were used only for writing assistance (grammar, phrasing) and did not contribute to the core methodology, experimental design, or analysis.

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.