

UltraQuant: 4-bit KV Caching for Context-Heavy Agents

Inesh Chakrabarti^{*,1,2} David Limpus^{*,1,3} Aditi Ghai Rana^{*,1}
Bowen Bao¹ Spandan Tiwari¹ Thiago Crepaldi¹ Ashish Sirasao¹

¹Advanced Micro Devices ²University of California, Los Angeles ³Purdue University

inesh33@ucla.edu dlimpus@purdue.edu
{aditi.rana, bowen.bao, spandan.tiwari}@amd.com
{thiago.crepaldi, ashish.sirasao}@amd.com

Abstract

Context-heavy agents place substantial pressure on the key-value (KV) cache: long prefixes are reused across many short turns, while concurrency determines whether the serving system can keep GPUs utilized. We study 4-bit KV-cache compression for this setting, using TurboQuant-style rotation and codebook quantization (Zandieh et al., 2026) as a quality anchor and vLLM FP8 KV caching as the deployment anchor. We report three contributions. First, we frame 4-bit KV caching around multi-round agent workloads where task quality, cache residency, and serving throughput must be measured jointly. Second, we describe the practical design choices needed to make the 4-bit path robust, including asymmetric K/V treatment, Walsh–Hadamard rotation (Walsh, 1923; Hadamard, 1893), QJL removal (Zandieh et al., 2024), and block-scale variants. Third, we present serving optimizations on AMD GPUs, including optimized decode-attention kernels and *UltraQuant*, an FP4 approximation path that uses FP8 queries, FP4 KV tensors, UE8M0 group scales, and native scaled-MFMA support on CDNA4. On an adaptive-SLO replay of production Claude Code traces, *UltraQuant* sustains $2.71\times$ (MiniMax-M2.5) and $4.38\times$ (Qwen3-235B) the qualified-request throughput of the BF16 baseline, matching or exceeding hardware FP8 KV while using half the KV bytes. *UltraQuant* delivers its largest gains in long-context, high-concurrency, memory-constrained serving regimes.

1 Introduction

Large language models have evolved from chatbots with short context windows into agents that browse the web, inspect repositories, invoke tools, and complete substantial software-engineering tasks (Yao et al., 2022; Schick et al., 2023; Zhou et al., 2023; Yang et al., 2024a). These workflows

require long-running memory: the model must retain system instructions, tool definitions, retrieved documents, codebase context, and the evolving plan across many turns. As model releases push context windows toward one million tokens and beyond (Gemini Team, 2024; Ding et al., 2024), the KV cache grows linearly with context length and becomes a first-order consumer of high-bandwidth memory (HBM).

This has driven two complementary lines of work: One line changes the model architecture or attention mechanism to reduce KV state directly, including multi-head latent attention (DeepSeek-AI, 2024a,b) and linear or state-space alternatives (Gu and Dao, 2023; Dao and Gu, 2024; Yang et al., 2024c,b). Another line treats the memory hierarchy as a systems problem, using cache reuse, offloading, and paged-memory management to keep scarce HBM occupied by the most useful state (Kwon et al., 2023; Zheng et al., 2023; Zhang et al., 2024). KV quantization adopts the first approach: we preserve the standard interface while reducing memory overhead.

Compression leads to some challenges; a deployable 4-bit method must offer more than just compression, but rather must retain answer quality on long-context workloads, integrate with paged serving, and dispatch efficiently through GPU matrix cores. FP8 KV caching is a strong deployment baseline in vLLM, giving roughly $2\times$ compression at near-lossless quality with native hardware support. TurboQuant (Zandieh et al., 2026), a data-oblivious 4-bit scalar-plus-rotation method, provides a strong 4-bit quality point but relies on codebook lookup and software dequant.

We study two endpoints on AMD Instinct GPUs. *Ultra-TQ* keeps the TurboQuant representation and closes the kernel gap through layout, lookup, and MFMA scheduling optimizations, along with accuracy improvements through calibrated centroids for each model. *UltraQuant* replaces the code-

^{*}Equal contribution.

book with an FP4 micro-tensor approximation so the dequantization is folded into native CDNA4 scaled-MFMA instructions.

Our goal is a deployment-oriented account of low-bit KV caching for context-heavy agents: which quantization choices preserve quality, how those choices interact with decode-attention kernels, and when the serving system benefits from the additional resident context. UltraQuant delivers its largest gains in long-context, high-concurrency, memory-constrained serving regimes.

2 Related Work

2.1 KV-Cache Quantization

The KV-compression landscape has converged on a few recurring design ideas. Keys and values are treated asymmetrically because key errors perturb the softmax distribution differently from value errors, a pattern established by KIVI (Liu et al., 2024) and KVQuant (Hooper et al., 2024). QJL (Zandieh et al., 2024) and TurboQuant (Zandieh et al., 2026) use rotation and codebook methods to improve rate-distortion at small bit widths, and TurboQuant in particular has seen community reimplementation (TheTom, 2026; Chakrabarti et al., 2026). However, the use of the codebook introduces lookup and irregular-access costs (Chakrabarti et al., 2026). These methods are useful algorithmic anchors for 4-bit quality, but their most accurate representations are not automatically efficient serving formats.

2.2 Rotation and Codebook Quantization

TurboQuant-style rotation (Kurtić et al., 2026) spreads outlier energy across channels so scalar quantizers see a more Gaussian-like distribution. Calibrated centroids fit via per-model Lloyd-Max (Lloyd, 1982; Max, 1960) or its per-block variant (LMPB) then provide a strong 4-bit codebook for the rotated distribution. In this paper, those codebooks are background and motivation rather than the deployed format: UltraQuant uses them to explain why FP4 is plausible after rotation, then replaces arbitrary centroids with the hardware-native FP4 grid.

2.3 Serving Systems and Hardware-Native Low Precision

vLLM-style paged serving (Kwon et al., 2023) makes KV-cache residency a systems concern, not only a compression metric. Hardware-native for-

mats such as FP8 already benefit from direct matrix-core support. UltraQuant extends this serving-first lens to 4-bit KV caching by targeting the CDNA4 scaled-F8F6F4 MFMA path, where FP4 operands and UE8M0 scales are consumed by the matrix core itself.

3 Agentic Serving on Production Traces

For context-heavy agents, each session keeps a long prefix resident in HBM across many turns, so the number of agents a GPU can serve under a latency service-level objective (SLO) depends on how much KV state fits in memory. A smaller KV budget fills first, evicts reusable prefix blocks, and forces expensive re-prefill that spikes time-to-first-token (TTFT). A KV method must therefore keep long prefixes resident at high concurrency and decode at near hardware-native speed, since dequantization overhead can offset the capacity it buys.

We emulate production Claude Code workloads using anonymized, production-derived sessions from the open-source kv-cache-tester corpus (Fox, 2024). Virtual users replay recorded sessions against our vLLM endpoint, reproducing multi-turn order, prompt/output-token targets, inter-turn think times, and prefix-sharing structure.

We measure serving under an adaptive-SLO protocol on MiniMax M2.5 and Qwen3-235B. An admission controller ramps from 6 to 48 concurrent sessions, gated by a rolling p95 TTFT against a 120-second target; admission pauses when TTFT headroom drops below the gate. This mimics how a production scheduler trades concurrency against latency. Our primary metric is TTFT-qualified goodput: the rate of completed requests whose first token arrives within the TTFT budget. We report cumulative SLO-qualified requests in the main text; Appendix E gives goodput curves and effective-TTFT tail distributions.

The key mechanism is capacity-driven prefix retention. FP8 KV ramps faster at low pressure but fills earlier, evicting reusable prefix blocks and forcing re-prefill. UltraQuant tracks FP8 KV decode speed under low pressure while offering $3.2\text{--}3.4\times$ BF16 logical KV capacity (vs. $2\times$ for FP8), reaching $2.34\times/3.25\times$ BF16’s prefix-hit ratio on MiniMax/Qwen (vs. $1.07\times/0.85\times$ for FP8). The avoided prefill work outweighs UltraQuant’s slight decode overhead under pressure.

Model	Scheme	Qual. @120s	Goodput (req/s/GPU)
MiniMax M2.5	BF16	77	0.064
	FP8 KV	168	0.140
	Ultra-TQ	140	0.117
	UltraQuant	209	0.174
Qwen3-235B	BF16	37	0.031
	FP8 KV	80	0.067
	Ultra-TQ	131	0.109
	UltraQuant	162	0.135

Table 1: TTFT-qualified goodput on the production-derived Claude Code adaptive trace replay (600 s assessment window, TP= 2, AMD MI355X). A completion qualifies if its effective TTFT is ≤ 120 s; goodput is the qualifying count divided by the assessment window and GPU count.

MiniMax M2.5. FP8 KV reaches the concurrency cap earlier, but UltraQuant’s SLO-qualified curve accelerates under pressure and finishes at $2.71\times$ BF16’s cumulative qualified-request total, compared with $2.18\times$ for FP8 KV and $1.82\times$ for Ultra-TQ (Fig. 1a).

Qwen3-235B. UltraQuant preserves more TTFT headroom as cache pressure rises, finishing at $4.38\times$ BF16’s qualified-request total vs. $2.16\times$ for FP8 KV and $3.54\times$ for Ultra-TQ (Fig. 1b).

Figures 1a and 1b normalize cumulative SLO-qualified turns by BF16’s qualified-request total. Across both models, FP8 KV has lower compute overhead at low pressure, while UltraQuant retains more KV state under high pressure. Once prefix replacement and re-prefill become expensive, UltraQuant’s capacity advantage outweighs its dequant cost. Table 1 reports the terminal TTFT-qualified goodput at the 120-second threshold on the same replay: UltraQuant reaches $2.71\times/4.38\times$ BF16’s qualified-request rate on MiniMax and Qwen. Aggregate throughput confirms this: UltraQuant reaches $1.25\times/1.29\times$ BF16’s input/output tok/s on MiniMax and $1.59\times/1.53\times$ on Qwen. Full TTFT-goodput curves and effective-TTFT tail panels for the same Claude Code replay are in Appendix E. The practical takeaway is that on identical hardware and under a fixed 120-second SLO, 4-bit KV serves $2.7\text{--}4.4\times$ more agent requests than BF16 while matching or exceeding hardware FP8 KV at half the cache footprint.

4 Ultra-TurboQuant

Ultra-TurboQuant (Ultra-TQ) is our optimized implementation of TQ_{4/4}, the 4-bit TurboQuant KV-cache scheme. It starts from the same observation that motivates TurboQuant: the raw KV-cache dis-

tribution is a poor target for a small scalar codebook, but a fixed Walsh–Hadamard rotation spreads outlier energy across channels, pushing each coordinate toward the light-tailed, near-Gaussian regime where a 4-bit scalar quantizer is a reasonable approximation. Appendix A presents efficient implementation details.

4.1 Calibrated Centroids

We begin with the observation that calibrating TurboQuant centroids is model agnostic. The production Lloyd–Max centroids assume post-rotation, post-normfold key coordinates follow $\text{Beta}((d-1)/2, (d-1)/2)$, the exact law of a randomly rotated unit-vector coordinate. On real activations this is only approximate, so the theoretical centroids are slightly misplaced. Refitting the 16-entry Lloyd–Max table on captured key activations corrects this. Calibration is cheap — a single forward pass over ~ 20 randomly sampled vectors from a sphere per rotated layer yields 16 FP32 levels — and deployment is a drop-in table swap with zero inference overhead. In practice, we only refit 10% of layers that show higher per-element quantization MSE.

Calibration is the strongest pure-codebook lever we measured: it lowers per-element K quantization MSE by 10.3% ($1.32 \times 10^{-4} \rightarrow 1.18 \times 10^{-4}$) and improves GPQA by +1.20 pp. The empirical centroids are compressed relative to the theoretical ones (largest $|c|$ at $d = 64$ falls from 0.342 to 0.288), reflecting the slight deviation of real activations from the ideal. Rotation remains essential: on the un-rotated distribution, even model-tuned centroids leave $\sim 2.3\times$ the rotated MSE, suggesting that centroid placement cannot replicate rotation’s whitening effect. The gain from calibrating centroids is real but modest; we ablate the source of the benefits in Appendix B.1.

5 UltraQuant

A calibrated codebook must be decoded at every attention step: each tile loads packed indices, gathers centroids through a lookup table, and reconstructs keys/values in registers before the QK and PV products (Guo et al., 2024). This gather is irregular and grows with context length. UltraQuant instead stores the cache as MXFP4: FP4 E2M1 codes with UE8M0 scales that the matrix core reads natively, making dequantization a single exponent shift folded into the instruction. The per-group scales is derived from one global constant, calibrated offline

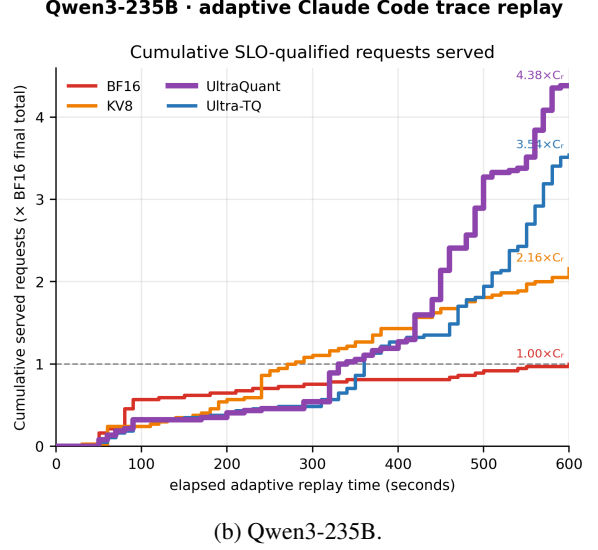
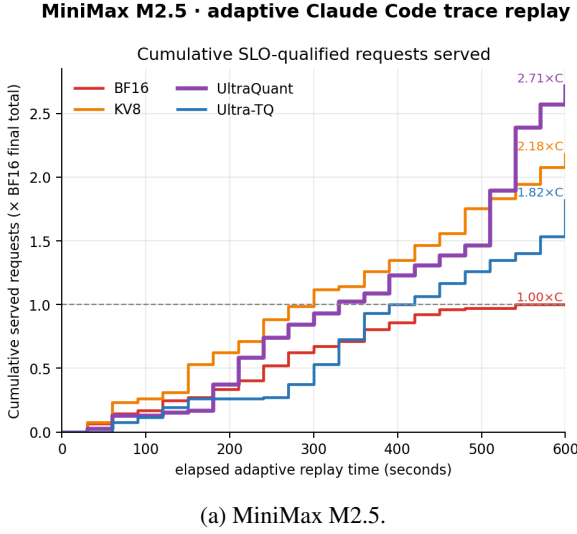


Figure 1: Cumulative SLO-qualified requests served on production-derived Claude Code traces (TP= 2, AMD MI355X), normalized relative to BF16.

and model-agnostic, that minimizes quantization error on the rotated distribution.

5.1 Cache Layout

UltraQuant stores keys/values in MXFP4. The head dimension is partitioned into contiguous groups of 32 channels, each encoded as 32 FP4 E2M1 codes (packed two per byte) plus one UE8M0 s-scale: $32 \times 4 \text{ bits} + 8 \text{ bits} = 17 \text{ B}$ per group, i.e. 4.25 bits per element — within 6% of an ideal 4-bit representation, in a format the matrix core consumes natively.

5.2 Dequantization

The dequantization rule is:

$$\text{value} = \text{code} \times 2^{\text{scale}}, \quad \text{scale} = \text{byte} - 127. \quad (1)$$

The scale is a power-of-two exponent that shifts the FP4 codepoint and folds directly into the scaled-MFMA accumulator. The AMD MFMA_SCALE_F32*_F8F6F4 instruction takes FP4 codes and the UE8M0 byte as native operands (Advanced Micro Devices, Inc., 2024), so keys and values are never materialized in BF16. Queries are Hadamard-rotated and rounded to FP8 E4M3, so attention runs on the $\text{FP8} \times \text{FP4}$ scaled-MFMA path. The constant c (Section 5.3) enters through the encoder rule $E = \text{round}(\log_2(c \cdot m))$ rather than as a runtime multiply.

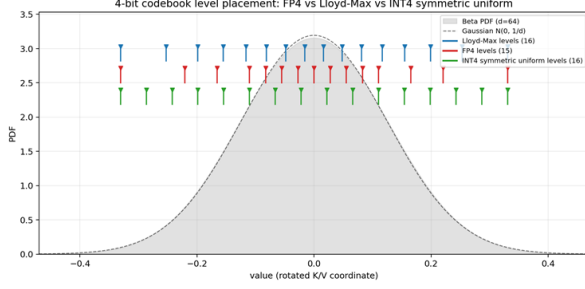
5.3 Constant-Optimized Scaling

Lloyd–Max centroids for the rotated distribution are symmetric, concentrated near zero, and increas-

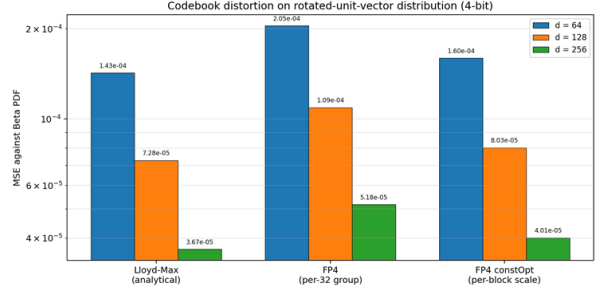
ingly wide in the tails: the pattern FP4 must approximate. Fig. 2a shows Lloyd–Max placing levels more densely in the high-probability body than uniform INT4, while FP4 E2M1 has fixed hardware-defined spacing. Plain per-group FP4 is consistently worse than Lloyd–Max in MSE (D’Albarto, 2026), but the constant-optimized per-block scale below recovers most of that gap while keeping the FP4 grid, as quantified in Fig. 2b and ablated in Appendix B.2.

Each group of 32 channels is mapped onto the FP4 grid by a single scale $s = c \cdot m$, where $m = \max_i |x_i|$ is the group’s absolute maximum and c is one global constant shared across every group, head, and model. We drop the per-token ℓ_2 normalization (K-norm) used by the codebook path: each block’s scale derives from its own absmax, making the blocks self-scaling. A per-token factor cannot be folded into the attention score (it does not pass through the softmax), so it would add a stored scalar and runtime multiply with no quality gain.

The per-group absmax m absorbs all per-token, per-head, and per-model magnitude variation: dividing by m maps each group into $[-1, 1]$. After Hadamard rotation the remaining *shape* is approximately $\text{Beta}((d-1)/2, (d-1)/2)$, depending only on head dimension d . This model-independence is why a single constant works: the one remaining degree of freedom is where this fixed shape sits against the E2M1 grid, i.e. the ratio $c = s/m$. In effect, the per-group absmax together with the constant c takes over the role of TurboQuant’s per-



(a) 4-bit codebook level placement at $d=64$.



(b) Codebook distortion (MSE).

Figure 2: Codebook analysis on the rotated-unit-vector distribution. (a) Lloyd–Max centroids provide the algorithmic reference, FP4 E2M1 is the hardware-native approximation, and symmetric INT4 is a uniform baseline. (b) FP4 has higher MSE than analytical Lloyd–Max, but constOpt per-block scaling closes much of the gap while preserving a hardware-native FP4 representation.

token K-norm.

We choose c to minimize the reconstruction error of the E2M1 grid. With $q(\cdot)$ the nearest-codepoint rounding onto $F = \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$:

$$c^* = \arg \min_c \mathbb{E} \left[\left(z - c m q(z/(cm)) \right)^2 \right]. \quad (2)$$

This one-dimensional minimization is solved once, offline, on captured rotated key activations. The minimizer is $c = 0.156$, deployed without per-model recalibration across MiniMax and Qwen (Section 6); it remains MSE-optimal in the single-model ablation in Appendix B.2. Figure 2b shows the codebook-level MSE gap.

This one-dimensional minimization is solved once, offline, on captured rotated key activations over a small calibration set. The minimizer on the captured keys is $c = 0.156$, the value we deploy. Note that the distribution post rotation is similar enough that this single fit transfers across models at a given head: we deploy the same $c = 0.156$, without any per-model recalibration, across the MiniMax and Qwen model families reported in Section 6, and it remains the MSE-optimal setting in the single-model ablation in Appendix B.2.

To ablate, we consider alternative constants in Appendix B.2; Figure 2b shows the corresponding codebook-level MSE gap.

6 Accuracy Results

All KV-compressed evaluations use boundary-layer protection: the first and last two attention layers retain BF16 KV caches ($n=2$). Our primary accuracy result is end-to-end agent task resolution on SWE-bench Lite (Section 6.1): it is the only benchmark here that grades a complete multi-turn agent loop rather than a single response, and it is the setting

Model	BF16	FP8 KV	Ultra-TQ	UltraQuant	Δ BF16
MiniMax M2.5	62	62	61	59	−3
Qwen3-235B	28	21	27	24	−4

Table 2: SWE-bench Lite resolved tasks (fixed 100-task denominator).

UltraQuant targets, since a long shared repository prefix with many follow-up turns is exactly where KV capacity binds. Sections 6.2 and 6.3 summarize retrieval and reasoning accuracy; their per-model tables and per-task breakdowns are in Appendix C.

6.1 SWE-bench Lite Agent Tasks

SWE-bench Lite evaluates end-to-end software-engineering task resolution: for each GitHub issue, an agent inspects the repository, edits over multiple turns, and submits a patch graded by project tests. We evaluate every KV format on a fixed 100-task cohort for MiniMax M2.5 and Qwen3-235B with prefix caching enabled (Table 2).

On MiniMax, UltraQuant resolves 59/100 tasks versus 62/100 for BF16, retaining 95% of BF16’s resolved-task count, with FP8 KV at 62/100 and Ultra-TQ at 61/100. On Qwen, UltraQuant resolves 24/100 versus 28/100 for BF16; it exceeds FP8 KV’s 21/100 and stays within three points of Ultra-TQ’s 27/100. Agent task resolution therefore lands within a few points of BF16 on both models while the KV cache is stored at 4 bits, and on Qwen both 4-bit schemes beat the 8-bit baseline despite its larger per-token budget. Figure 19 in Appendix C plots resolved-task counts for the same cohort under all four KV schemes.

6.2 Retrieval Tasks

Retrieval accuracy is largely preserved and the residual loss is task-dependent rather than uniform. On Llama-3.1-8B-Instruct, UltraQuant matches

BF16 on NIAH and on the 13-task LongBench-E average. Using NVIDIA’s official RULER suite (4K–131K, four tasks), Qwen3.5-35B-A3B reaches exact parity at 99.38, while MiniMax M2.5 gives up ground at 74.03 versus 67.08, with the loss concentrated in `single_2` and `multikey_1` and a gain on multiquery (Appendix C).

6.3 Reasoning Tasks

We evaluate every KV format on GPQA Diamond, GSM8K, and AIME25. Deltas are small in both directions: UltraQuant stays within 2 points of BF16 on every model–benchmark pair except Qwen3-32B GPQA Diamond, where it gains 5.58 points. On MiniMax M2.5 GPQA Diamond the intermediate schemes fall between the two endpoints, with FP8 KV at 83.33 and Ultra-TQ matching BF16 at 83.84 (Fig. 17). MiniMax M2.5 AIME25 is averaged over five seeds; Llama-3.1-8B-Instruct AIME25 is omitted for insufficient coverage.

7 Performance Results

We measure UltraQuant’s decode throughput and per-token latency against four baselines: vLLM OSS TurboQuant (Kurtić et al., 2026), BF16 AITER (AMD ROCm, 2025) FlashAttention, hardware FP8 KV, and Ultra-TQ, on MiniMax-M2.5, TP= 2, 2×MI355X, 32K/1K at concurrency 64. All results are reported relative to the BF16 AITER FlashAttention baseline (= 1.00×). Folding dequantization into the MFMA gives UltraQuant nearly the same instruction mix as the FP8 kernel, so it tracks the FP8 throughput ceiling while halving the KV-cache footprint. Figure 3 reports throughput across the kernels: at $C=64$ UltraQuant reaches 1.38× the BF16 output throughput, matching hardware FP8 KV (1.37×) to within ~1%, while serving twice the KV footprint per unit of HBM.

The FP4 path largely removes software dequantization from the critical path while retaining the memory-capacity benefit of a 4-bit KV cache. UltraQuant delivers 1.38× the BF16 baseline throughput, on par with the hardware FP8 KV-cache path (1.37×) while using half the KV bytes per element. On per-token latency, BF16 has no dequant, so all 4-bit kernels sit above it; UltraQuant’s median TPOT is 1.40× BF16, within ~2% of the hardware FP8 KV-cache path (1.37×) and well ahead of Ultra-TQ (1.58×) and vLLM OSS TurboQuant (5.56×) (Fig. 4a).

The advantage grows with context length. Fig-

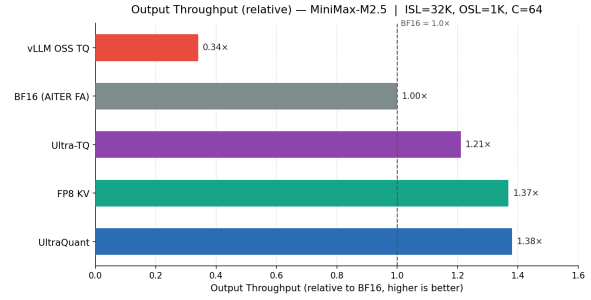


Figure 3: UltraQuant throughput (relative to BF16) vs. BF16, FP8 KV, and Ultra-TQ on MiniMax-M2.5 (TP= 2, 2×MI355X).

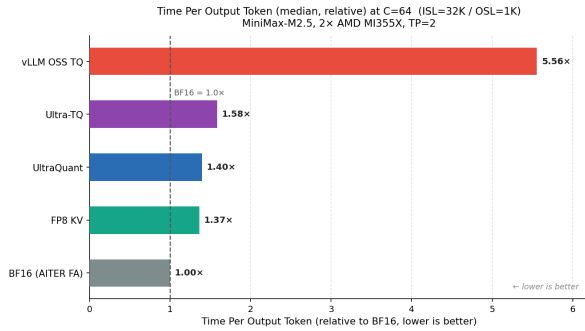
ure 4b sweeps inter-token latency against input context from 8K to 64K: UltraQuant improves monotonically relative to BF16, reaching ~0.5× at 64K, while FP8 stays near 1.3–1.5× and Ultra-TQ only catches up at the longest contexts. The half-precision KV footprint is what lets UltraQuant sustain longer contexts before HBM pressure forces re-prefill.

7.1 Sensitivity to Concurrency, Input Length, and Output Length

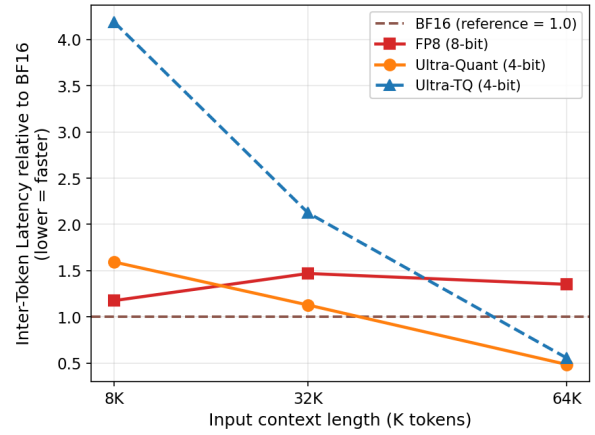
The $C=64$ results above fix concurrency, input length, and output length simultaneously. We additionally vary each axis independently on MiniMax-M2.5, holding the others fixed. UltraQuant tracks FP8 KV closely on raw throughput and tail latency as concurrency and output length grow. At long input context, FP8 KV hits its resident-cache memory wall around 40K context and begins evicting/recomputing sequences, while UltraQuant throughput and TTFT stay flat through the same range: 4-bit KV buys capacity, not kernel speed, and serves the long-context load that makes FP8 KV fall behind. The same input-length sweep on Qwen3-235B-A22B shows a consistent trend. Full per-axis figures, discussion, and captions are in Appendix B.4.

7.2 Kernel-Level Roofline Analysis

Two mechanisms explain why UltraQuant tracks FP8 on raw throughput and latency while pulling ahead at long context and high load. First, 4-bit KV approximately doubles nominal cache capacity relative to FP8, reducing eviction and re-prefill under pressure (Section 7.1). Second, long-context decode is bandwidth-bound, so reading 4-bit KV moves about half as many bytes; at short context or decode-heavy configurations, FP8 can be faster because the dequantization overhead is not yet amortized.



(a) Median time per output token, $C=64$.



(b) Inter-token latency vs. input context length.

Figure 4: Per-token latency on MiniMax-M2.5 (32K/1K, TP= 2, 2×MI355X), relative to BF16 (lower is better) for FP8, UltraQuant, and Ultra-TQ.

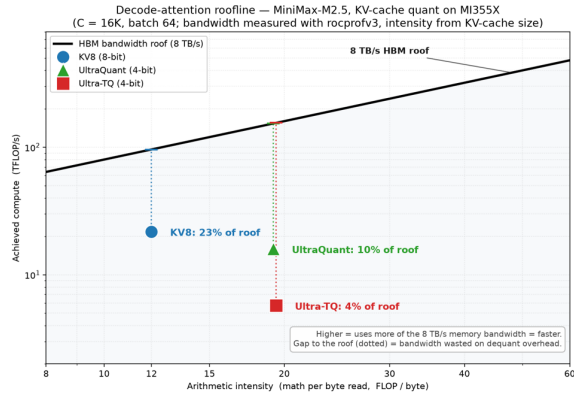


Figure 5: Decode-attention roofline on MiniMax-M2.5 (MI355X, $C=16K$, batch 64): achieved compute vs. arithmetic intensity for FP8 KV, UltraQuant, and Ultra-TQ against the 8 TB/s HBM bandwidth roof.

Figure 5 makes the kernel-level mechanism explicit with a decode-attention roofline on MiniMax-M2.5 (MI355X, $C=16K$, batch 64). The horizontal axis is arithmetic intensity (math per byte read); FP8 KV sits furthest left because 8-bit KV moves the most data per unit of math, while UltraQuant and Ultra-TQ shift right by $\sim 1.6\times$ rather than the nominal $2\times$, since metadata and padding leave a real footprint of about 5 effective bits per element. The vertical axis is achieved TFLOPs against the bandwidth-implied compute ceiling: FP8 KV reaches 23% of that ceiling, UltraQuant reaches 10%, and Ultra-TQ reaches only 4%.

The winning condition is reaching 4-bit intensity while staying close to bandwidth-limited rather than dequant-limited: UltraQuant reaches 4-bit intensity and stays near bandwidth-bound, $\sim 2.5\times$ higher than Ultra-TQ on the roofline, whose software codebook decode is dequant-bound. Kernel

counters at 32K context confirm UltraQuant retains most of the memory saving at near-FP8 kernel time; the remaining $\sim 18\%$ gap is why its largest end-to-end wins come from capacity and avoided re-prefill (Section 7.1) rather than a universal per-token speedup. Appendix A.3 gives the per-scheme breakdown and counter table.

8 Conclusions

We presented a 4-bit KV-cache study centered on context-heavy agents, where long prefixes must remain reusable and concurrency determines GPU utilization. TurboQuant-style rotation and codebook methods provide a strong 4-bit quality anchor; Ultra-TQ closes the gap to production serving through careful layout, LUT, and MFMA scheduling; and UltraQuant replaces the codebook with MXFP4 tensors consumed directly by CDNA4 matrix cores. The broader lesson is that low-bit KV caching should be evaluated as an end-to-end serving mechanism, not only as an offline compression method. For agent workloads, the relevant outcome is the joint effect on task success, cache residency, throughput, and interactivity. Our results show that 4-bit KV caching retains the context-capacity benefits of aggressive compression while approaching the deployability of hardware-native FP8 KV caching.

9 Limitations

We present the three following limitations:

1. Although we deploy a single constant $c = 0.156$ for all models and heads, there may be a more efficient constant that can be calibrated

layer-wise for better accuracy. We also do not consider the case that the constant is chosen depending on the distribution of activations. We omit this out of simplicity and leave this for future work.

2. UltraQuant’s benefits in speed versus FP8 are only observed when the context length is long enough, or concurrency high enough, to exceed the resident cache capacity of the FP8 baseline; for shorter context lengths and lower concurrency, the benefits of the algorithm are not realized. Note that this is not the case when compared to Ultra-TQ, where performance benefits are always realized. Larger clusters, other serving frameworks, and non-CDNA4-optimized performance are also outside the scope of this evaluation.
3. Ultra-TQ style centroid calibration is only applied to 10% of layers for a small benefit; on most layers, such centroid calibration does not seem to provide any benefit.

References

- Advanced Micro Devices, Inc. 2024. *AMD Instinct CDNA4 Instruction Set Architecture*.
- AMD ROCm. 2025. *AITER: AI tensor engine for ROCm*. AMD ROCm Blogs.
- Inesh Chakrabarti, David Limpus, Aditi Ghai Rana, Bowen Bao, Spandan Tiwari, Thiago Crepaldi, and Ashish Sirasao. 2026. *Productionizing TurboQuant on AMD GPUs for KV-cache-bound LLM inference*. AMD ROCm Blogs.
- Paolo D’Alberto. 2026. *Statistical inference and quality measures of KV cache quantisations inspired by TurboQuant*. *Preprint*, arXiv:2605.08114.
- Tri Dao and Albert Gu. 2024. *Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality*. *Preprint*, arXiv:2405.21060.
- DeepSeek-AI. 2024a. *DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model*. *Preprint*, arXiv:2405.04434.
- DeepSeek-AI. 2024b. *DeepSeek-V3 technical report*. *Preprint*, arXiv:2412.19437.
- Yiran Ding, Li Lina Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. *LongRoPE: Extending LLM context window beyond 2 million tokens*. *Preprint*, arXiv:2402.13753.
- Callan Fox. 2024. *kv-cache-tester: Production-derived trace replay for KV cache serving evaluation*. GitHub repository.
- Gemini Team. 2024. *Gemini 1.5: Unlocking multi-modal understanding across millions of tokens of context*. *Preprint*, arXiv:2403.05530.
- Albert Gu and Tri Dao. 2023. *Mamba: Linear-time sequence modeling with selective state spaces*. *Preprint*, arXiv:2312.00752.
- Han Guo, William Brandon, Radostin Cholakov, Jonathan Ragan-Kelley, Eric P. Xing, and Yoon Kim. 2024. *Fast matrix multiplications for lookup table-quantized LLMs*. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12419–12433.
- Jacques Hadamard. 1893. *Résolution d’une question relative aux déterminants*. *Bulletin des Sciences Mathématiques*, 17:240–246.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. *KVQuant: Towards 10 million context length LLM inference with KV cache quantization*. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Eldar Kurtić, Michael Goin, and Alexandre Marques. 2026. *A first comprehensive study of TurboQuant: Accuracy and performance*. vLLM Blog. Red Hat AI.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. *Efficient memory management for large language model serving with PagedAttention*. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*.
- Fan Li, Siyuan Feng, Chen Huang, Dake Wang, Hongbin Yang, Peng Sun, and Emad Barsoum. 2026. *Fly-DSL: Expert GPU kernel development with the ease of MLIR Python native DSL on AMD GPUs*. AMD ROCm Blogs.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. *KIVI: A tuning-free asymmetric 2bit quantization for KV cache*. In *International Conference on Machine Learning (ICML)*.
- Stuart P. Lloyd. 1982. *Least squares quantization in PCM*. *IEEE Transactions on Information Theory*, 28(2):129–137.
- Joel Max. 1960. *Quantizing for minimum distortion*. *IRE Transactions on Information Theory*, 6(1):7–12.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. *Toolformer: Language models can teach themselves to use tools*. *Preprint*, arXiv:2302.04761.

TheTom. 2026. [TurboQuant+: community llama.cpp port and exploration of TurboQuant](#). GitHub repository.

Joseph L. Walsh. 1923. [A closed set of normal orthogonal functions](#). *American Journal of Mathematics*, 45(1):5–24.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024a. [SWE-agent: Agent-computer interfaces enable automated software engineering](#). *Preprint*, arXiv:2405.15793.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. 2024b. [Gated delta networks: Improving Mamba2 with delta rule](#). *Preprint*, arXiv:2412.06464.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. 2024c. [Parallelizing linear transformers with the delta rule over sequence length](#). *Preprint*, arXiv:2406.06484.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. [ReAct: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.

Amir Zandieh, Majid Daliri, Mojtaba Hadian, and Vahab Mirrokni. 2026. [TurboQuant: Online vector quantization with near-optimal distortion rate](#). In *International Conference on Learning Representations (ICLR)*.

Amir Zandieh, Majid Daliri, and Insu Han. 2024. [QJL: 1-bit quantized JL transform for KV cache quantization with zero overhead](#). *Preprint*, arXiv:2406.03482.

Yichi Zhang, Bofei Gao, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, Wen Xiao, and 1 others. 2024. [PyramidKV: Dynamic KV cache compression based on pyramidal information funneling](#). *arXiv preprint arXiv:2406.02069*.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2023. [SGLang: Efficient execution of structured language model programs](#). *Preprint*, arXiv:2312.07104.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. [WebArena: A realistic web environment for building autonomous agents](#). *Preprint*, arXiv:2307.13854.

A Kernel Implementation in vLLM

We implement two serving endpoints in vLLM: *Ultra-TQ*, an optimized TurboQuant kernel that preserves the codebook representation, and *Ultra-Quant*, an FP4 approximation path in which the KV cache is stored as raw FP4 micro-tensors with

Algorithm 1 FP4-ConstOpt KV encoding (Ultra-Quant)

Require: K, V (BF16); rotation R ; group size $g=32$; E2M1 levels F ; global constant c ([Section 5.3](#)).

Write:

- 1: $K \leftarrow R \cdot K$ $\triangleright$ rotate keys; V unrotated
- 2: split K, V into 32-channel groups
- 3: **per group:** $m \leftarrow \max_i |x_i|$; $E \leftarrow \text{round}(\log_2(c \cdot m))$ $\triangleright$ UE8M0 exponent; no per-token norm, no search
- 4: **per coord:** $\text{code} \leftarrow q(x_i/2^E)$ $\triangleright q$: round to nearest level in F
- 5: store 2 FP4 codes/byte + 1 UE8M0 byte (E) per group

Decode:

- 6: $Q \leftarrow R \cdot Q$; $Q \leftarrow \text{cast_fp8_e4m3}(Q)$
- 7: load codes + UE8M0 byte; $\hat{x} \leftarrow \text{code} \cdot 2^{(\text{byte}-127)}$
- 8: QK via scaled FP8 $\times$ FP4 MFMA; PV via FP4/scale path

Output: hardware-native FP4+UE8M0 KV cache; dequant folds into the MFMA scale operand.

UE8M0 group scales and the dequantization is performed by the matrix core itself.

The implementation targets the CD-NA4 scaled-MFMA family, especially `v_mfma_scale_f32_16x16x128_f8f6f4`. The QK product reads FP4 keys, FP8 queries, and UE8M0 scales together in one instruction. The V path similarly avoids a software LUT by converting FP4 codes and UE8M0 scales through hardware-supported low-precision operations. Relative to a TurboQuant codebook kernel, the key systems change is removing dequantization from the critical path: the query operand is pre-rounded to FP8 outside the K-tile inner loop, and the grouped scale layout used by the cache is arranged to match the MFMA scale operand layout. [Algorithm 1](#) gives the UltraQuant cache-encoding path; the codebook-based Ultra-TQ encoding it replaces is in [Appendix F](#).

A.1 Ultra-TQ Kernel Ladder

TurboQuant compresses the KV cache to 4 bits per element, storing keys and values in a codebook format. The cache is dequantized on the fly, making decode memory-bandwidth-bound. Our kernels are inspired by FLUTE ([Guo et al., 2024](#)), which restructures quantized matrices offline to

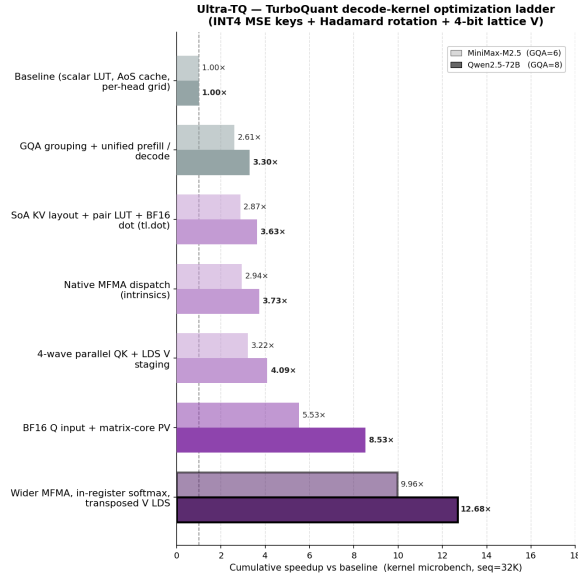


Figure 6: Ultra-TQ optimization ladder for the decode-attention kernel.

reduce bit-manipulation overhead during unpacking and uses vectorized LUT access to reduce shared-memory pressure. We adapt these ideas to the vLLM decode-attention path by adding GQA-aware tiling, a structure-of-arrays KV cache layout, and AMD-specific MFMA dispatch.

The Ultra-TQ optimization ladder has three tiers. First, an improved Triton kernel fixes the GQA, LUT, and layout pathologies of the open-source TurboQuant baseline. Second, a native-ISA implementation issues matrix-core instructions through AMD GCN intrinsics, giving direct control over operand layout and instruction selection. Third, a FlyDSL (Li et al., 2026) implementation JIT-compiles to AMD GCN ISA without per-device source files and exposes MFMA operand layouts so we can select wider MFMA variants.

A.2 UltraQuant Kernel Ladder

The UltraQuant kernel takes the FP4 plus UE8M0 cache layout described in Section 5 and adds three CDNA4-native operations to the same D-SL implementation. First, the QK product is issued through the scaled F8F6F4 MFMA (`v_mfma_scale_f32_16x16x128_f8f6f4`), which reads FP4 keys, FP8 queries, and their UE8M0 scales together in one instruction; BF16 keys never have to be materialized. Second, the V-dequant LUT is replaced by a hardware conversion path that turns FP4 codes and UE8M0 scales directly into BF16. This removes software dequantization from the critical path. Third, the FP8 query operand is lifted out of the K-tile inner loop so every tile reuses

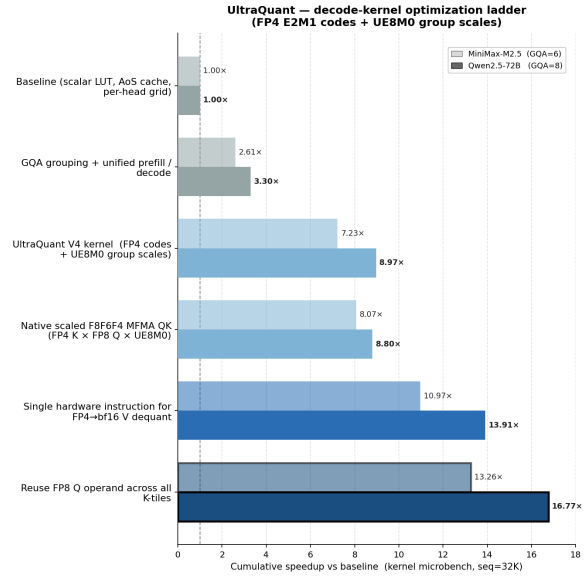


Figure 7: UltraQuant optimization ladder for the FP4 decode-attention kernel.

the same pre-quantized query, reducing redundant packing and easing register pressure.

The implementation runs in the standard ROCm/vLLM stack. Ultra-TQ’s FlyDSL decode kernel has been upstreamed into vLLM mainline. We plan to release the UltraQuant kernel as a follow-up contribution.

A.3 Decode-Attention Kernel Counters

This supplements the roofline analysis in Section 7.2. FP8 KV is bandwidth-bound and closest to the roof, but requires $2\times$ the KV-cache space, halving the available capacity before eviction. Ultra-TQ reaches 4-bit intensity but is dequant-bound: its limiter is software codebook decode, an LDS gather, and a VALU FMA per element, so in the worst case the compute cost outweighs the bandwidth saving. UltraQuant reaches 4-bit intensity and stays closer to bandwidth-bound, $\sim 2.5\times$ higher than Ultra-TQ on the roofline, because FP4 E2M1 with UE8M0 scales is consumed natively by the scaled MFMA with no codebook, gather, or per-element FMA.

Table 3 reports the corresponding decode-attention kernel counters at 32K context, batch 8. The counters confirm that 4-bit KV roughly halves HBM traffic; Ultra-TQ’s software LUT decode is compute-bound despite the smaller payload, while UltraQuant folds dequantization into the CDNA4 scaled-MFMA path, retaining most of the memory saving at near-FP8 kernel time.

KV format	Width	HBM bytes/step	Decode attn. (32K, batch 8)
FP8 KV	8-bit	~23 MB	1071 ms (1.00 $\times$)
Ultra-TQ	4-bit	~12 MB	2624 ms (2.45 $\times$)
UltraQuant	4-bit	~12 MB	1268 ms (1.18 $\times$)

Table 3: Decode-attention kernel comparison at 32K context, batch 8, MiniMax-M2.5 on MI355X. UltraQuant nearly halves HBM traffic relative to FP8 KV while staying within 1.18 $\times$ its kernel time; Ultra-TQ’s software dequant erases the bandwidth saving.

Configuration	Adaptation	Codebook	GPQA
TQ-t4nc (production)	per-token ℓ_2	Lloyd (calib.)	0.6503
K+V Lloyd, per-token	per-token ℓ_2	Lloyd (calib.)	0.6237
LMPb full	per-block absmax	Lloyd (calib.)	0.6559
Variant E	per-block absmax	uniform (RTN)	0.6528

Table 4: Per-block scale ablation (GPT-OSS-20B GPQA, medium, single seed). Holding rotation on, the per-token $\rightarrow$ per-block adaptation moves 0.6237 $\rightarrow$ 0.6559 on the both-Lloyd configs; with per-block scaling fixed, Lloyd versus uniform codebook (0.6559 vs. 0.6528) is within noise. Codebook and adaptation are co-fit, so the swap is not a perfectly clean A/B, but the direction is consistent across variants.

B Ablations

B.1 Per-Block Scale

This ablation isolates the contribution of the per-block-absmax scale from that of the calibrated codebook, on GPT-OSS-20B GPQA (medium effort, single seed; $1\sigma \approx 1.7$ pp). Two effects stand out. First, switching the adaptation statistic from a single per-token ℓ_2 norm to a per-block absmax (groups of 32) is the load-bearing change: it is what recovers accuracy, independent of the codebook. Second, with the per-block scale in place the codebook itself barely matters — replacing the calibrated Lloyd magnitudes with a uniform grid stays within single-seed noise. Together, these justify the deployed design: keep the per-block scale, and use the fixed FP4 grid in place of a calibrated, model-specific codebook.

B.2 Global Constant

Figure 2b quantifies the codebook approximation: plain per-32-group FP4 sits above the analytical Lloyd–Max floor, but the constOpt per-block scale closes most of that MSE gap while keeping the hardware-native FP4 grid, across $d \in \{64, 128, 256\}$.

We also sweep the global scaling constant c against the FP8 (8-bit) baseline, on the same evaluation. The default $c = 0.156$ is the MSE-optimal value derived in Section 5.3, and the data confirm it: it beats FP8 by +4.4 pp. Raising c to 0.195 (the power-of-two-optimal value, see Section 9) gives

Scheme	Accuracy	vs. FP8 baseline
FP8 (8-bit baseline)	63.0%	—
fp4 $c = 0.156$ (default)	67.4%	+4.4 pp
fp4 $c = 0.195$	63.0%	0.0 pp
fp4 $c = 1.0$ (no shrinkage)	58.7%	−4.3 pp

Table 5: Global-constant ablation. The default $c = 0.156$ is MSE-optimal and the only setting that improves on FP8; larger c erases the gain and $c = 1.0$ (raw absmax) regresses below the 8-bit baseline.

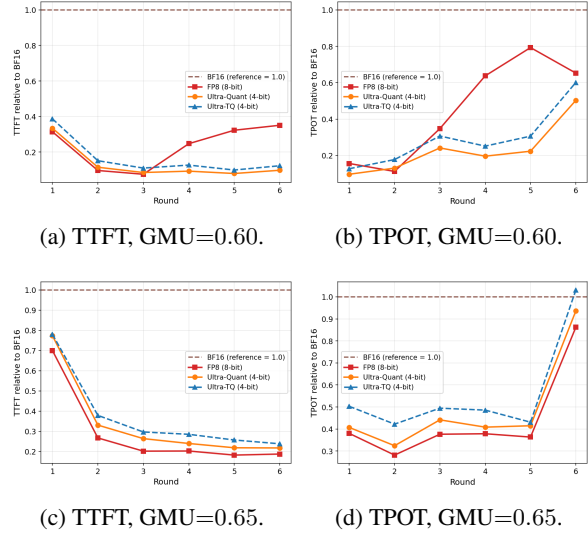


Figure 8: Per-round serving latency (relative to BF16; lower is better) at two cache-pressure regimes. Top row: GMU=0.60 (the Appendix D operating point), where FP8 degrades in later rounds. Bottom row: GMU=0.65, where the larger resident-cache budget keeps all three schemes close.

up the gain entirely and lands exactly at the FP8 baseline, and $c = 1.0$ (raw absmax with no MSE shrinkage) falls 4.3 pp below FP8. Accuracy is monotonic in the distance from the MSE-optimal constant.

B.3 Cache-Pressure (GMU) Regime

We report the per-round latency at two GPU-memory-utilization (GMU) operating points to show how the relative ranking shifts with cache pressure. Figure 8 stacks both regimes: the top row is GMU=0.60 (the operating point used in Appendix D) and the bottom row is the lower-pressure GMU=0.65. At GMU=0.60 the resident-cache budget is tight, so FP8 degrades in the later rounds while UltraQuant stays low; at GMU=0.65 the budget is large enough that FP8 no longer degrades and all three schemes track closely. The multi-turn ablation in Appendix D uses the more cache-pressured GMU=0.60 regime, where the UltraQuant residency advantage is largest.

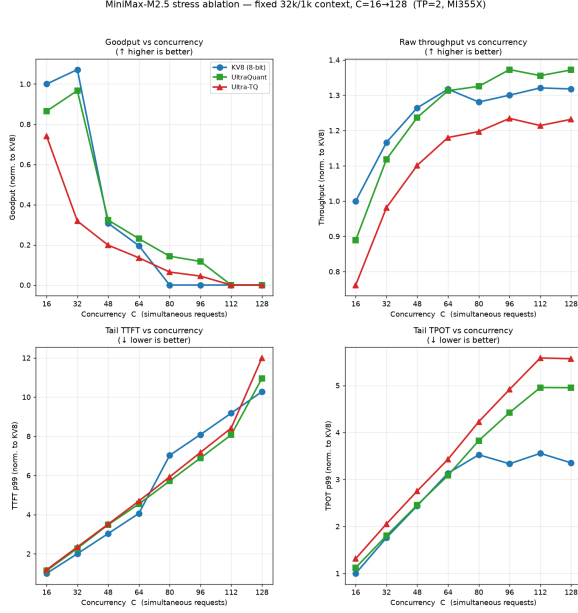


Figure 9: Full concurrency sweep on MiniMax-M2.5 (TP= 2, 2xMI355X): goodput (top-left), raw throughput (top-right), P99 TTFT (bottom-left), and P99 TPOT (bottom-right) vs. load C for FP8 KV, UltraQuant, and Ultra-TQ.

B.4 Concurrency, Input-Length, and Output-Length Sensitivity

This section gives the full panel-level detail for the sensitivity sweeps summarized in Section 7.1, on MiniMax-M2.5 (TP= 2, 2xMI355X).

Figure 9 sweeps concurrency C . Goodput (top-left) falls sharply for all schemes as load rises, with UltraQuant sustaining it marginally longer than FP8 KV. Raw throughput (top-right) is close to tied between FP8 KV and UltraQuant, with Ultra-TQ trailing by $\sim 10\%$ due to software dequant. P99 TTFT (bottom-left) rises roughly linearly with C and the three schemes overlap. P99 TPOT (bottom-right) is the clearest separation: FP8 KV’s tail TPOT plateaus because it caps admitted batch size under queuing pressure, while the 4-bit schemes admit more concurrent requests and pay a higher per-token cost, climbing to ~ 1.5 – $1.6\times$ that plateau.

Figure 10 sweeps input context length at fixed load. At short/medium context, FP8 KV is fastest and the 4-bit schemes pay a small tax (UltraQuant ~ 1 – 3% , Ultra-TQ ~ 7 – 10%). At 40K context, FP8 KV hits its resident memory wall: it can no longer keep 64 long sequences resident and begins dropping/recomputing them. UltraQuant’s throughput pulls ahead past this point, and FP8’s TTFT explodes to $\sim 7\times$ its short-context level while UltraQuant stays flat, $\sim 5\times$ faster to first token at the same setting. 4-bit KV buys capacity, not kernel

Benchmark	BF16	UltraQuant	Δ BF16
Official RULER	74.03	67.08	−6.95
GPQA Diamond	83.84	81.82	−2.02
GSM8K	88.86	87.11	−1.74
AIME25 (5 seeds)	79.17	78.33	−0.84

Table 6: MiniMax M2.5 accuracy (%).

Benchmark	BF16	UltraQuant	Δ BF16
Official RULER [†]	99.38	99.38	0.00
GPQA Diamond [‡]	51.01	56.57	+5.58
GSM8K [‡]	95.53	95.22	−0.30
AIME25 [‡]	16.67	16.67	0.00

Table 7: Qwen accuracy (%). [†]Qwen3.5-35B-A3B. [‡]Qwen3-32B.

speed: it serves the long-context load that makes FP8 KV fall behind.

Figure 11 fixes the prompt and grows the answer length, stressing pure decode. Throughput (left) rises and plateaus as the fixed prefill cost is amortized over more output tokens; UltraQuant shadows FP8 KV to within 2–3% at every output length. TPOT (middle) starts higher and settles to a steady rate once decoding is in full swing, with UltraQuant sitting a few percent above FP8 KV at a constant gap. TTFT (right) is flat and identical across all three schemes, confirming output length affects decoding only.

We also ran the input-length ablation on Qwen3-235B-A22B (Fig. 12, TP= 4, fixed output length 1K, $C=64$). It shows the same trend as MiniMax-M2.5 in Fig. 10: UltraQuant keeps up with FP8 KV on decode throughput, time-to-first-token, and per-token latency across context lengths, while using half the KV memory.

C Additional Accuracy Results

This appendix supplements Section 6 with the per-model accuracy tables, per-task retrieval breakdowns, extended model validation, and SWE-bench Lite cohort plots omitted from the main text for space.

Tables 6 to 8 report BF16 versus UltraQuant accuracy by model family, with benchmarks as rows, for the retrieval and reasoning results discussed in Sections 6.2 and 6.3. Official RULER on MiniMax M2.5 is the largest single regression at -6.95 ; every other cell sits within 2 points of BF16 in either direction.

Figure 13 compares KIVI and official KVQuant against BF16 and UltraQuant on Llama-3.1-8B-Instruct NIAH and LongBench-E. KIVI tracks

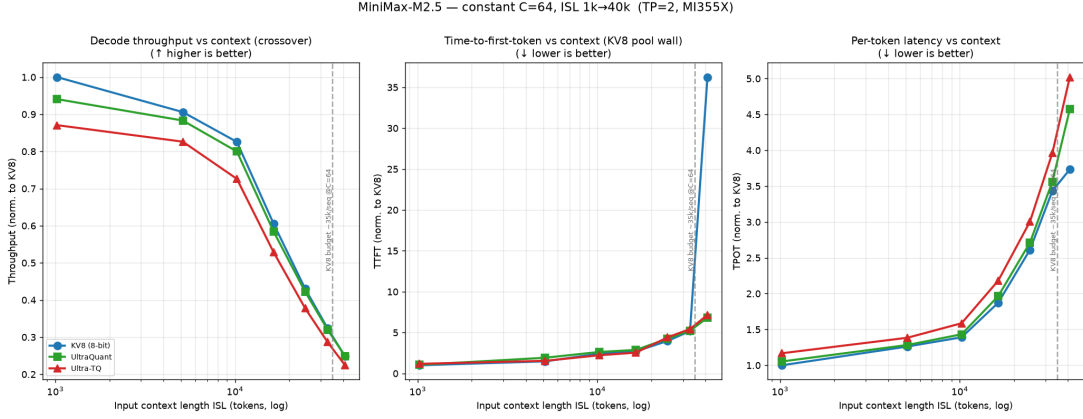


Figure 10: Full input-length sweep at fixed load on MiniMax-M2.5 (TP= 2, 2×MI355X): throughput and TTFT for FP8 KV, UltraQuant, and Ultra-TQ across input context lengths.

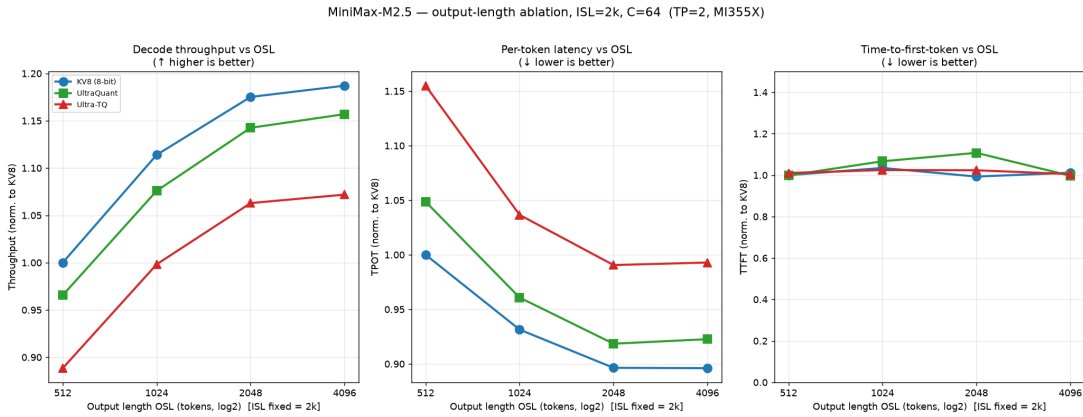


Figure 11: Full output-length sweep at fixed prompt size on MiniMax-M2.5 (TP= 2, 2×MI355X): throughput (left), TPOT (middle), and TTFT (right) for FP8 KV, UltraQuant, and Ultra-TQ.

Benchmark	BF16	UltraQuant	Δ BF16
NIAH	99.5	100.0	+0.50
LongBench-E (13 tasks)	54.19	53.97	-0.22
GPQA Diamond	29.29	27.27	-2.02
GSM8K	78.09	77.03	-1.06

Table 8: Llama-3.1-8B-Instruct accuracy (%).

BF16 through 65K context but OOMs at 131K; channel-wise runtime quantization remains difficult because it quantizes across sequence length during decode.

Figure 14 breaks down the official RULER retrieval subset for Qwen3.5-35B-A3B (four tasks $\times$ six lengths $\times$ 15 examples per method), where UltraQuant reaches exact parity with BF16 at 99.38% in aggregate (Table 7).

Figure 15 gives the same RULER breakdown for MiniMax M2.5; loss concentrates in single_2 and multikey_1, with a gain on multiquery, matching the 74.03 vs. 67.08 aggregate in Table 6.

Figure 16 reports RULER for Llama-3.1-8B-Instruct, including official KVQuant, contextualizing the NIAH and LongBench-E summaries in Table 8.

Figure 17 complements Tables 6 to 8 with per-benchmark BF16 versus UltraQuant panels on G-PQA Diamond, GSM8K, and AIME25; on MiniMax GPQA the intermediate schemes sit between the two reported endpoints (FP8 KV at 83.33, Ultra-TQ matching BF16 at 83.84).

Figure 18 validates Qwen3.6-27B on GSM8K (5-shot) and LiveCodeBench-128K; error bars are ± 1 standard error.

Figure 19 plots resolved-task counts for MiniMax M2.5 and Qwen3-235B under all four KV schemes on the fixed 100-task SWE-bench Lite cohort summarized in Table 2.

D Multi-Turn Workload Ablation

Beyond the production Claude Code replay in Section 3, we corroborate the same capacity effect on a multi-turn workload that lets us set memory pressure directly. We use the Gutenberg text corpus, from which vLLM synthesizes multi-turn conversations in user/assistant role format under configurable prefix and turn-length distributions. We run on MiniMax M2.5 and Qwen3-235B at TP= 2 on

Qwen3-235B-A22B (FP8) — context-length ablation, OSL=1k, C=64 (TP=4, MI355X)

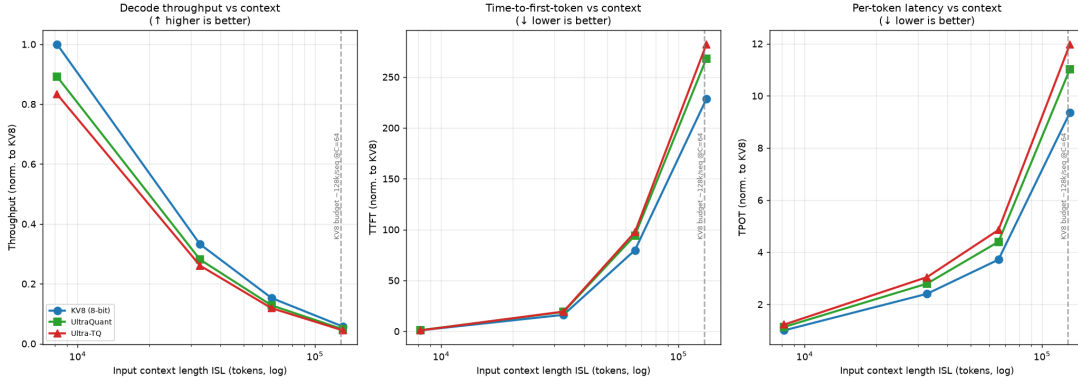


Figure 12: Input-length (context-length) sweep on Qwen3-235B-A22B (TP= 4, fixed output length 1K, $C=64$): decode throughput, TTFT, and per-token latency vs. context length for FP8 KV, UltraQuant, and Ultra-TQ.

Current reportable KV-cache method comparisons

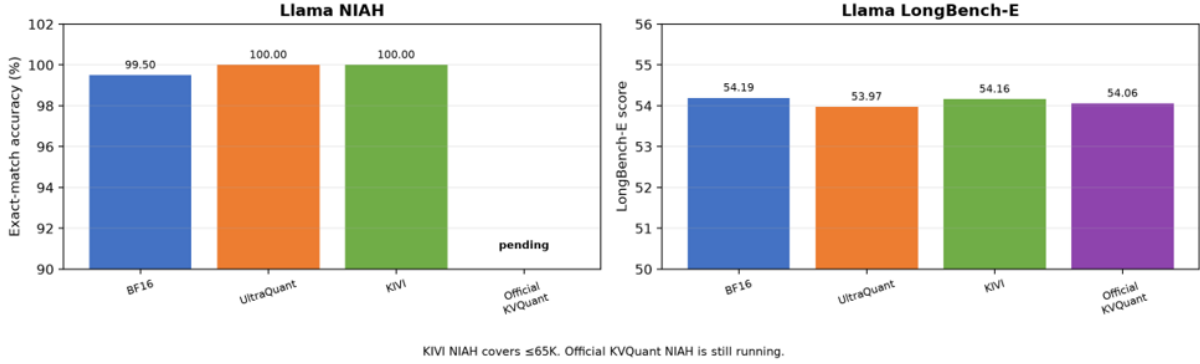


Figure 13: Llama-3.1-8B-Instruct NIAH and LongBench-E accuracy versus KIVI and official KVQuant. KIVI NIAH covers $\leq 65K$ context.

AMD MI355X, fixing concurrency at $C=32$ and reporting P50 TTFT and P50 TPOT per round.

Table 9 and Fig. 20 report this workload on MiniMax M2.5. With long per-client prefixes and high concurrency, FP8 KV repeatedly evicts useful s-state while UltraQuant keeps more prefixes resident. The advantage is largest in the late rounds, where P50 TTFT improves $3.47\times$: UltraQuant holds both TTFT and TPOT low across all six rounds, while FP8 degrades as accumulated context exceeds its resident-cache capacity.

E Claude Code Adaptive SLO Supplement

This appendix supplements Section 3 with the remaining adaptive-SLO panels from the production-derived Claude Code trace replay. Figure 21 reports TTFT-qualified completed-request goodput: the rate of finished requests whose first token arrives within each swept TTFT budget, normalized by the BF16 qualified-request total. Figure 22 reports effective-TTFT tail distributions

Metric	UltraQuant vs. FP8 KV
P50 TTFT — warm rounds (r2–3)	$0.86\times$ (FP8 faster)
P50 TTFT — late rounds (r4–6)	$3.47\times$
P50 TTFT — all rounds	$2.3\times$
Output throughput	$1.63\times$

Table 9: Gutenberg multi-turn serving results on MiniMax-M2.5, TP= 2, AMD MI355X, reported as UltraQuant relative to the FP8 KV baseline (higher is better except where noted). The advantage appears in the late rounds, where long per-client prefixes exceed the effective resident-cache capacity of FP8: TTFT improves $3.47\times$ and is recovered through cache residency rather than re-prefill.

(p50/p90/p95/p99) at the same operating points. On MiniMax M2.5, UltraQuant reaches $2.71\times$ BF16 on both cumulative SLO-qualified served requests (main text) and TTFT-qualified goodput; on Qwen3-235B, both metrics reach $4.38\times$ BF16.

Qwen3.5-35B-A3B: official RULER retrieval subset
Aggregate — BF16 99.38, UltraQuant 99.38

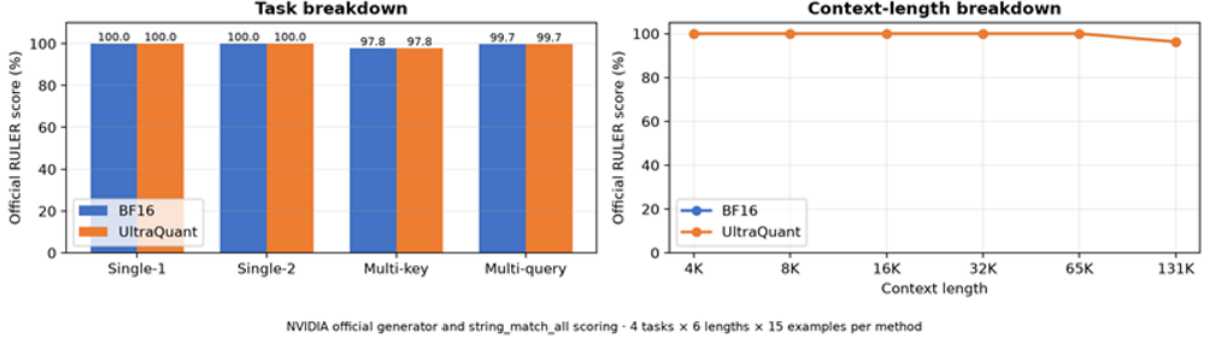


Figure 14: Official RULER retrieval subset for Qwen3.5-35B-A3B (four tasks × six lengths × 15 examples per method).

MiniMax-M2.5: official RULER retrieval subset
Aggregate — BF16 74.03, UltraQuant 67.08

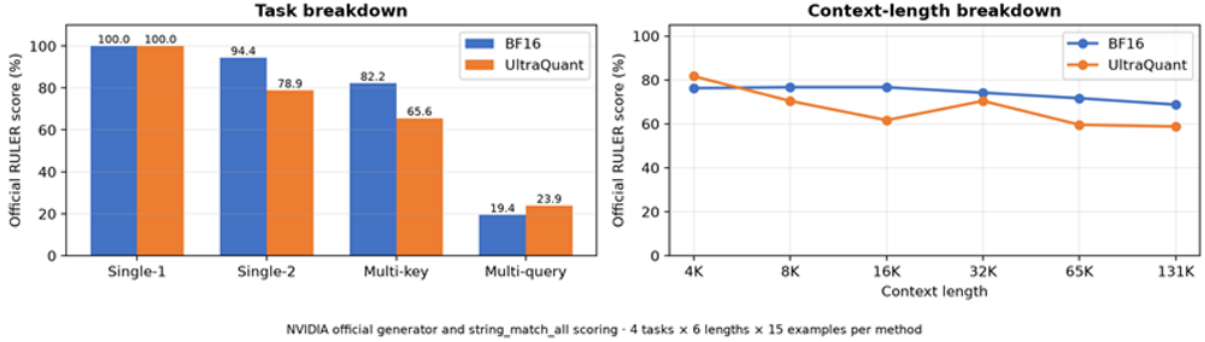


Figure 15: Official RULER retrieval subset for MiniMax M2.5.

F Calibrated LUT int4 KV Encoding (Ultra-TQ)

For reference, [Algorithm 2](#) gives the codebook-based encoding path that UltraQuant replaces. It is the TurboQuant-style scheme: a calibrated 4-bit codebook per side with a per-token norm scalar on the key side, dequantized through a lookup table on read. UltraQuant ([Algorithm 1](#)) keeps the rotation but drops the codebook and the per-token norm in favor of a fixed FP4 grid with a per-block UE8M0 scale, so the dequantization folds into the matrix core.

Algorithm 2 Calibrated LUT int4 KV encoding (Ultra-TQ)

Require: K, V (BF16); rotation R ; group size g ; codebook C_K .

Write:

- 1: $K \leftarrow R \cdot K$ ▷ rotate keys; V unrotated
- 2: split K, V into g -channel blocks
- 3: K : store block norm; encode each coord to nearest C_K index
- 4: V : store block (scale, zero); encode each coord to 4-bit uniform

Decode:

- 5: $Q \leftarrow R \cdot Q$
- 6: load K/V codes + metadata per tile
- 7: K : dequant via $C_K \cdot \text{norm}$; V : dequant via $\text{scale} \cdot \text{code} + \text{zero}$
- 8: compute QK and PV on dequantized K, V

Output: rotated-codebook K + uniform V cache; dequant paid on read.

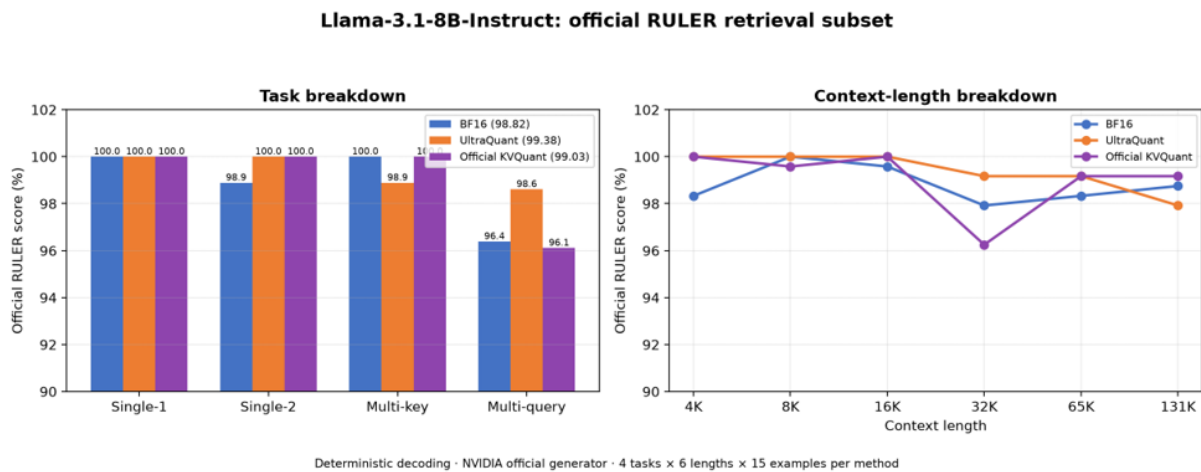


Figure 16: Official RULER retrieval subset for Llama-3.1-8B-Instruct, including official KVQuant.

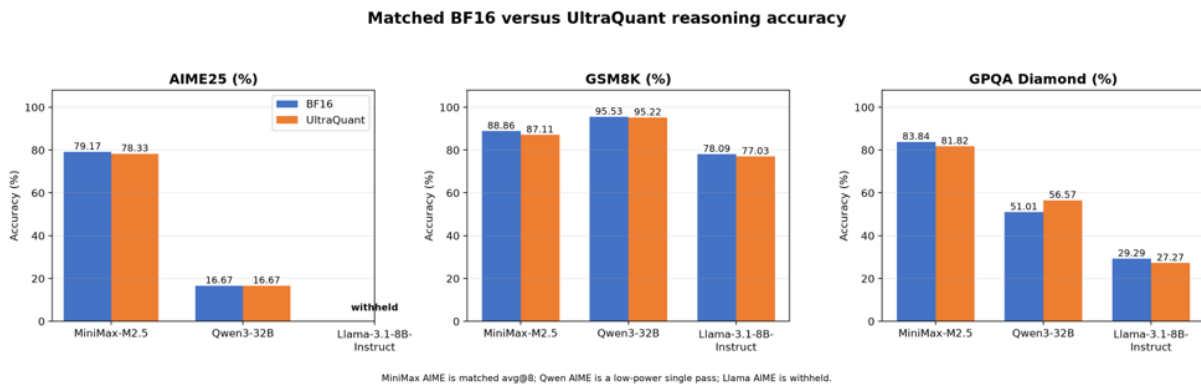


Figure 17: BF16 versus UltraQuant accuracy on GPQA Diamond, GSM8K, and AIME25 (Tables 6 to 8).

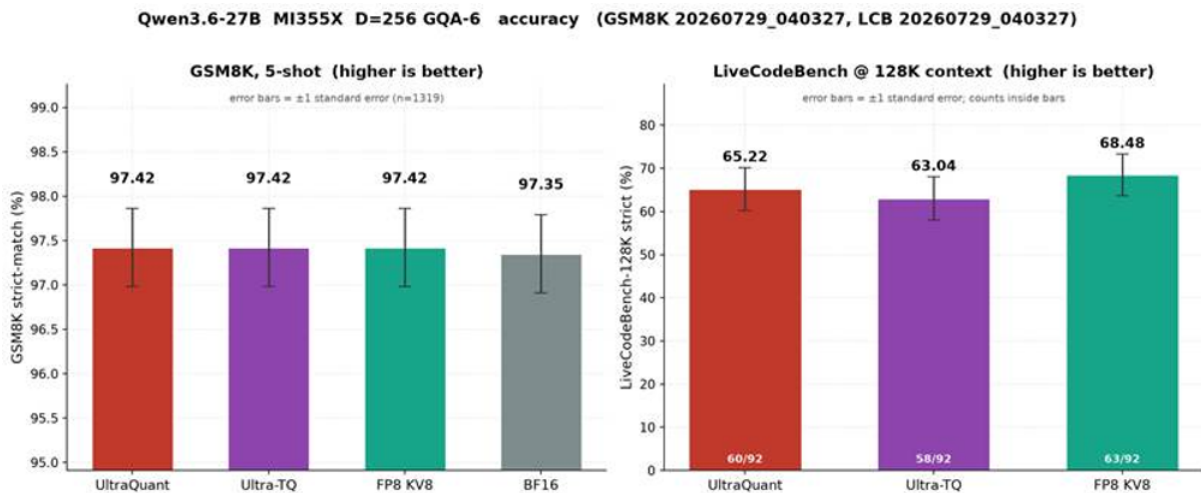


Figure 18: Qwen3.6-27B GSM8K (5-shot) and LiveCodeBench-128K accuracy on MI355X (TP= 2). Error bars are ±1 standard error.

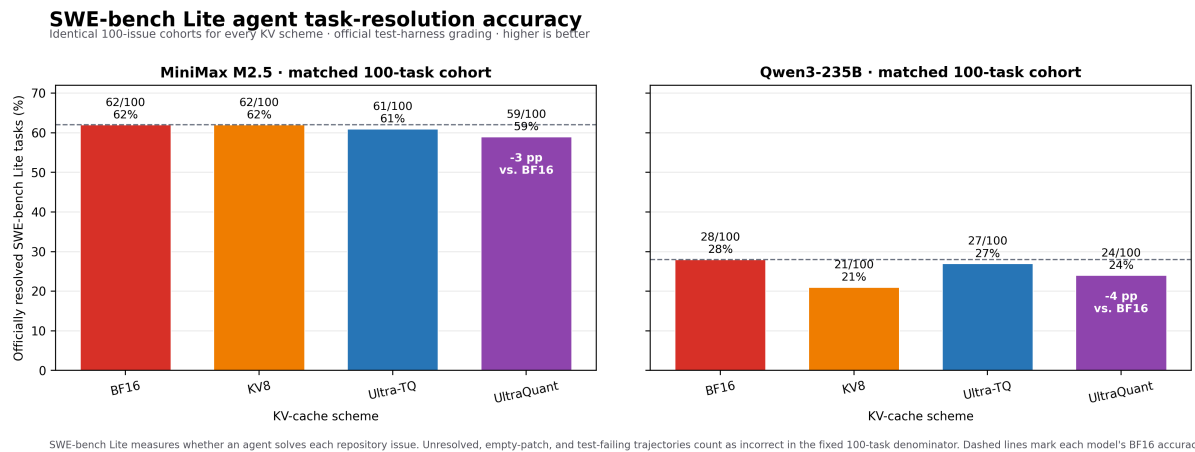


Figure 19: SWE-bench Lite agent task-resolution accuracy on fixed 100-task cohorts for MiniMax M2.5 and Qwen3-235B (Table 2).

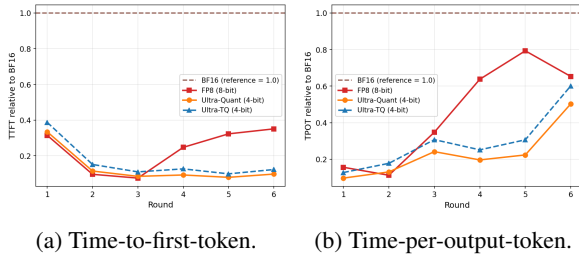
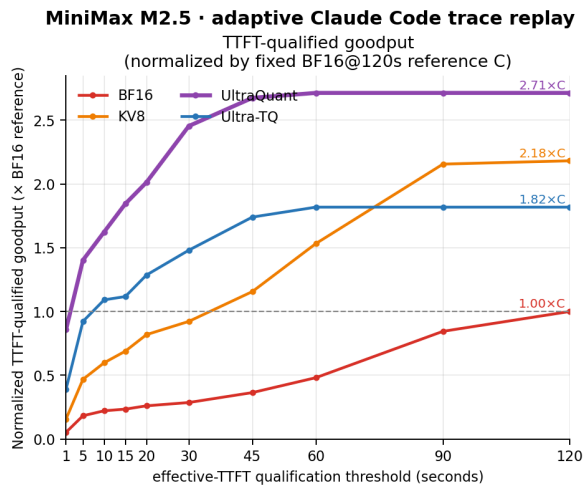
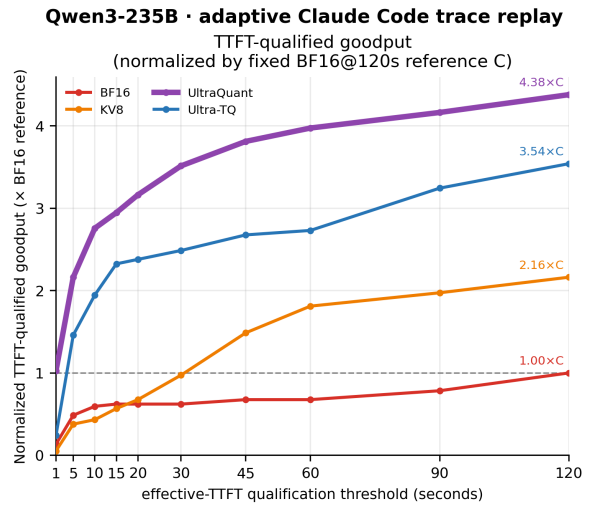


Figure 20: Per-round serving latency (relative to BF16; lower is better) on the Gutenberg multi-turn workload for FP8, UltraQuant, and Ultra-TQ. UltraQuant keeps prefixes resident, so both (a) time-to-first-token and (b) time-per-output-token stay low across all rounds, while FP8 degrades as accumulated context exceeds its resident-cache capacity.

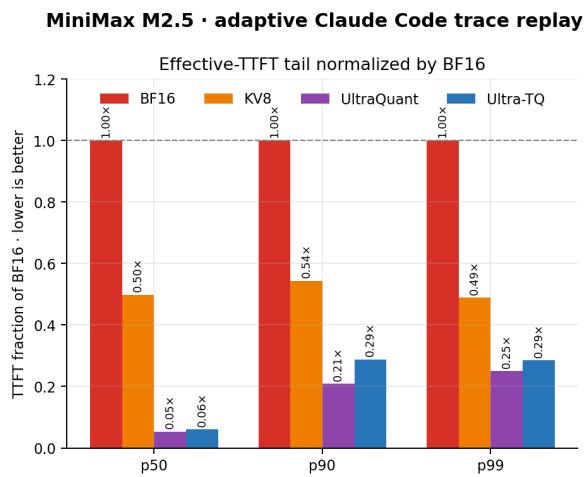


(a) MiniMax M2.5.

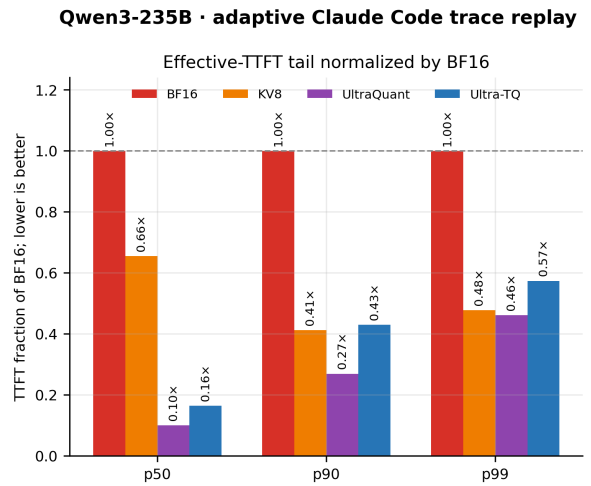


(b) Qwen3-235B.

Figure 21: TTFT-qualified completed-request goodput on the Claude Code trace replay (TP= 2, AMD MI355X), normalized relative to BF16.



(a) MiniMax M2.5.



(b) Qwen3-235B.

Figure 22: Effective-TTFT tail distributions (p50/p90/p95/p99) on the Claude Code trace replay (TP= 2, AMD MI355X).