

# When Does KV Cache Compression Pay? A Quality- and Cost-Aware Runtime Framework for Cloud LLM Serving

Anonymous Author(s)

Anonymous Institution

**Abstract.** KV cache compression makes two promises to a distributed LLM serving system: a compressed cache preserves model output, and shipping fewer bytes is faster. This paper shows that both promises fail unless checked against measured quantities. Fewer bytes are not automatically faster: codec, metadata, and consumer overhead can dominate small transfers, so a compressed path beats fp16 transport only above a link- and size-dependent crossover, and on a measured same-GPU path it never pays at all. Fewer bytes are not automatically safe either: aggressive quantization can violate a quality target — uniform INT4 collapses a 3B model while remaining PPL-neutral at 125M scale — so the admissible compression ratio is model-dependent. We therefore propose a measurement-calibrated runtime framework that decides *when* compression is worth deploying by accounting quality, latency, and memory jointly: (i) an adaptive INT4/INT8/FP16 codec enforces a writer-verified per-write reconstruction budget, checked on the actual packed representation at every write; (ii) a fixed-cost-aware transport ledger accounts codec, metadata, transfer, and consumer costs and yields a closed-form crossover; (iii) a profile-based deployment resolver returns the highest-compression feasible operating point or explicit infeasibility from measured profiles and operator constraints. The resolver selects certified  $B=0.2$  under a 1% PPL-degradation SLO and  $B=0.5$  under 3%, admits a uniform-INT4 point only for the smallest model at a 5% SLO, and prefers fp16 on the same-GPU path. The strongest result is negative: *compression may not pay* — its benefit is link- and size-dependent, model-dependent KV magnitudes bound the achievable ratio, and no single compression ratio is universally deployable.

**Keywords:** KV cache compression, LLM serving, quantization, disaggregated inference, deployment decisions

## 1 Introduction

KV cache compression makes two promises to a distributed LLM serving system. The quality promise is that a compressed cache preserves model output; the latency promise is that shipping fewer bytes is faster. Both promises are kept weakly in current systems: accuracy is *measured after the fact* on a held-out corpus, and the latency benefit is computed from a byte-only bandwidth model

that ignores the codec. This paper asks the deployment question directly: *when does KV cache compression actually pay* — under which quality targets, on which link, at which chunk size, and for which model? Our measurements show it pays much less often than the byte-count argument suggests.

*Fewer bytes are not automatically faster.* The compressed path carries fixed costs the byte-only model omits: quantize, metadata, and consumer overhead. Under our transport ledger, these fixed costs dominate below a link-dependent crossover  $S_{16}^*$ : on the measured 192-cell grid on an idle A100, every compressed path has  $S_{16}^* = \infty$  on our same-GPU device-to-device proxy — compression *never* pays there — while on PCIe the dequantize-then-use paths cross over at 1.06–2.08 MB, inside the 256 KB–4 MB range disaggregated serving uses (Sec. 4.6).

*Fewer bytes are not automatically safe.* The admissible compression ratio is model-dependent: uniform INT4 is PPL-neutral at 125M scale (+2.20%), degrades measurably at 1B scale (+9.41%), and *collapses* a 3B model ( $\Delta\text{PPL} \approx 12462\times$ ), because per-block KV magnitudes grow with scale and carry proportional reconstruction error (Sec. 4.3). A fixed compression ratio is therefore not universally deployable, and a per-write check, not another offline measurement, is what makes a deployment decision trustworthy.

We build a *measurement-calibrated runtime framework* that answers the deployment question with three components. The operator states quality, latency, and memory constraints; measured profiles — an error curve per model, a transport grid per link, a byte ledger per codec — determine feasibility: today a ratio  $r$  is chosen and quality is measured once, offline, with no runtime evidence and a byte-only latency model; our framework resolves the operator’s constraints through measured codec, transport-ledger, and per-write-verification profiles into a feasible operating point or explicit infeasibility.

### Contributions.

- **A writer-verified runtime invariant.** An adaptive INT4/INT8/FP16 codec enforces a sequence-wide per-element reconstruction budget at write time, verified on the actual packed representation (Sec. 3.3) and compositional over immutable KV writes (Lemma 2); an input-space guarantee, not an attention-output bound.
- **A fixed-cost-aware transport ledger.** Codec, metadata, transfer, and consumer costs are accounted jointly (Sec. 3.1); the closed-form crossover (3) shows the certified path beats fp16 transport only above a link- and size-dependent threshold —  $\infty$  on the measured same-GPU proxy, 1.06–2.08 MB on PCIe, 23–46 MB on a real dual-GPU P2P link.
- **A measured-profile deployment resolver.** Given measured codec profiles and operator constraints, the resolver (Sec. 3.5) returns the highest-compression feasible point or explicit infeasibility, validated by live vLLM and dual-GPU experiments (Secs. 4, 5).

The remainder presents related work (Sec. 2), the framework (Sec. 3), the evaluation (Sec. 4), the serving integration (Sec. 5), limitations (Sec. 6), and conclusions (Sec. 7).

## 2 Related Work

Three literatures meet in this paper: rate-distortion KV compression (the *a-posteriori* quality line), error-bounded lossy compression (the *by-construction* line), and disaggregated KV transport (the cost line). Our contribution is the deployment question at their intersection: given a codec’s *measured* error curve, transport grid, and byte ledger, when is the compressed path deployable?

*Rate-distortion KV compression (a posteriori error).* The quantization line — SmoothQuant [14], KVQuant [5], KIVI [12], QServe [8] — compresses KV with low-precision codecs and reports *measured* perplexity or downstream degradation; token-eviction schemes make the same epistemic move at a different granularity. All of these improve the ratio-accuracy frontier; the error is established *a posteriori* on a corpus, not verified against a user-specified bound on the sequence being served. Our complement is a write-time check on the compressed representation itself, and — equally — an accounting of whether the checked path is worth its overhead.

*Error-bounded lossy compression (by-construction error).* SZ [3], ZFP [9], and MGARD [1] bound per-element error by construction on dense scientific arrays; these codecs target dense field data, not token-major attention tensors, and their guarantees are stated per tensor with no composition story for an autoregressive sequence. Recent KV adaptations exist (PackKV [6]): those schemes construct bounds offline and validate them on a corpus. We add the serving half of that story — per-write verification of the bound on the actual reconstruction, and the ledger question (does the certified path beat fp16 transport?), which error-bounded codecs have never been asked. Sec. 3.6 asks it of PackKV directly: the released compressor, compiled from source and measured on our testbed, enters the same resolution through the same three curves.

*Disaggregated KV transport (the cost ledger).* Disaggregated serving (Splitwise [13]) decouples prefill and decode and moves KV chunks over a link; chunk sizes of 256 KB–4 MB are the operating regime. CacheGen [10] and HACK [15] explicitly address compressed KV transmission and its execution overheads: CacheGen adapts KV encoding and streaming to network conditions, while HACK avoids explicit dequantization through computation on quantized KV. Bandwidth analyses in this setting typically compare byte counts and ignore the fixed codec cost — quantize kernel, dequantize/unpack kernel, scale metadata bytes, consumer GEMM — which dominates small messages; our ledger quantifies this “fixed-cost tax,” and the resulting crossover is link-dependent (Secs. 3.4, 4.6).

*Recent runtime-aware KV systems.* Recent work has moved beyond static compression: KVServe [11] performs service-aware runtime profile selection for communication-efficient disaggregated serving; Runtime-Certified Bounded-Error Quantized Attention [2] and WitCert [7] provide runtime bounds closer to attention/output behavior; SplitZip [4] targets high-throughput lossless KV transport. Our scope is narrower and complementary: we study how measured

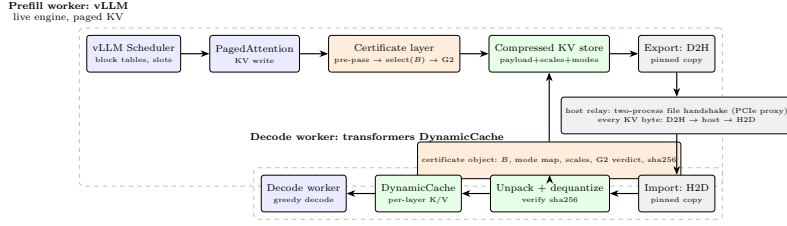


**Fig. 1.** Serving architecture with the certified-codec layer. **Prefill worker** (a live vLLM engine): paged-attention KV writes pass through the certificate layer — per 128-element block the pre-pass computes the candidate INT4/INT8 errors, the mode is selected under the budget  $B$ , and gate G2 verifies the actual reconstruction before the write commits — and are packed into the certificate envelope: payload, per-block fp32 scales, mode map,  $B$ , the G2 verdict, and a payload digest. The envelope crosses transport (the host-relay PCIe proxy of Sec. 5.3, or a dual-GPU PCIe P2P link) to the **decode worker**, which verifies the digest, dequantizes into its cache, and decodes without computing prefill attention.

representation quality, verification/codec overhead, transport cost, and memory footprint jointly determine whether a compressed path is deployable. The reconstruction certificate used here is an input-space writer-side invariant, not an attention-output guarantee.

### 3 Framework: Ledger, Codec, and Deployment Resolver

We consider the disaggregated decode path (Fig. 1): a KV chunk  $\mathbf{X} \in \mathbb{R}^{n_{\text{tok}} \times d_{\text{kv}}}$  (bf16,  $d_{\text{kv}} = 2048$  default) must move from the prefill worker to the decode worker over a link. The chunk is laid out *token-major*: row  $t$  is the KV of token  $t$ , and each row is partitioned into *per-token blocks* of  $\text{BLK} = 128$  contiguous elements — the layout consumed by the attention GEMM, with the flat per-block scale array aligned to the GEMM’s K-groups.

#### 3.1 System Model and Byte Ledger

Per-path end-to-end time  $T_{\text{path}}$  decomposes as

$$T_{\text{path}} = \alpha_{\text{codec}} + \beta_{\text{link}} \cdot \text{bytes\_moved} + T_{\text{consumer}}, \quad (1)$$

where  $\alpha_{\text{codec}}$  is the fixed codec cost (quantize, dequantize/unpack, launch overheads — the “codec tax”  $\Delta\alpha \equiv \alpha_{\text{codec}} - \alpha_{16}$  over the fp16 path’s fixed overhead  $\alpha_{16}$ ),  $\beta_{\text{link}}$  is the per-byte transport slope of the link, and  $T_{\text{consumer}}$  is the attention-proxy GEMM following decompression. **Byte accounting is complete:** payload bytes, fp32 per-block scale bytes ( $4 \cdot n_{\text{el}}/G$ ,  $G \in \{64, 128\}$ ), and — for the adaptive codec — one mode byte per block are all counted, and the PCIe leg (counted twice, host round trip) versus the device-to-device leg (counted once) are distinguished; the baseline is fp16 transport with no codec. Timings follow one measurement rule: per-repetition CUDA-event timing, median of  $\geq 50$  repetitions, warmup, sanity floors, a noise guard (coefficient of variation  $< 15\%$ ), an environment guard that flags co-tenant compute, and per-case correctness gates G1–G6; any violation invalidates the run.

### 3.2 Error-Bounded Adaptive Codec

For a block with values  $x_{b,i}$  ( $i = 1..\text{BLK}$ ) and magnitude  $m_b = \max_i |x_{b,i}|$ , the codec family is per-block max-scale symmetric quantization with scale  $s_w = m_b/(2^{w-1} - 1)$  and quantization rule  $q = \text{clamp}(\text{round}(x/s_w), \pm(2^{w-1} - 1))$ , giving worst-case per-element error  $e_8 \leq m_b/254$  and  $e_4 \leq m_b/14$ , so that the pre-pass, the quantize kernel, and the dequantize kernel replicate each other’s arithmetic exactly. One pre-pass kernel computes the two candidate errors  $e_8(b)$ ,  $e_4(b)$  per block by running the actual quantize/dequantize round trip at block scope (gate G1 checks the pre-pass against the codec, tolerance  $10^{-3}$ ). Given the user-specified budget  $B$ :

$$\text{mode}(b) = \begin{cases} \text{INT4}, & e_4(b) \leq B, \\ \text{INT8}, & e_8(b) \leq B, \\ \text{FP16 bypass}, & \text{otherwise,} \end{cases} \quad (2)$$

where the FP16 bypass stores the block bit-exactly (gate G3). The mixed-width kernels pack the selected modes with compacted per-block byte slots; mode bytes are part of the transport ledger. Gate G6 checks the monotonicity of the selection in  $B$ . The global budget is a *single knob*: the per-block decision makes the scheme automatically per-(layer, position) adaptive.

### 3.3 Writer-Verified Reconstruction Invariant

**Definition 1 (Writer-verified per-write ( $L_\infty$ ) reconstruction budget).** Fix a sequence-wide per-write ( $L_\infty$ ) reconstruction budget  $B > 0$  and a verification slack  $\eta = 10^{-3}$ . A KV chunk written at decode step  $t$  is certified with respect to  $B$  iff, on the actual reconstruction  $\hat{\mathbf{X}}_t$  from its stored (mixed-width) representation,  $\max_{x \in \mathbf{X}_t} |x - \hat{x}| \leq B(1 + \eta)$ ; the certificate  $\mathcal{C}_t(B)$  is the claim, checked at write time, that this inequality holds for the chunk. The guarantee is sequence-wide under immutable writes:  $\forall t : |\mathbf{X}_t - \hat{\mathbf{X}}_t|_\infty \leq B(1 + \eta)$ . It constrains the live KV writes of the actual trajectory only — never a hypothetical full-fp16 trajectory — and does not bound attention outputs.

**Trust model.** The writer holds the pre-compression KV and is trusted to execute the verification gate correctly. The certificate is intended to detect implementation drift, arithmetic mismatch, layout errors, and mode-selection violations before a compressed chunk is committed. The receiver can validate payload integrity but cannot independently recompute the reconstruction bound without the original KV. The mechanism is therefore a writer-verified runtime invariant, not a cryptographic proof against a malicious writer.

After every write, gate G2 computes the maximum absolute error on the *actual reconstruction* and fails the write if it exceeds  $B(1 + \eta)$  — separating the scheme from both the a-posteriori line (no bound is ever verified) and the SZ line (bounds validated offline, not per write); on violation the write falls back to fp16 for the offending blocks.

**Lemma 1 (Construction).** Under the selection rule (2), every block of a certified chunk satisfies  $\max_i |x_{b,i} - \hat{x}_{b,i}| \leq B$ , and gate G2 passes on the actual reconstruction.

*Proof.* By case analysis on (2): mode INT4 implies per-element error  $\leq e_4(b) \leq B$ ; mode INT8 implies error  $\leq e_8(b) \leq B$ ; FP16 bypass has zero error.

**Lemma 2 (Composition over immutable KV writes).** (i) *Block bounds compose: a token’s KV (the union of its per-token blocks), a layer chunk, and the union over all writes of a sequence each inherit maximum absolute error  $\leq B(1+\eta)$ .* (ii) *Each element’s error is determined and verified at the moment the element is written; stored KV is never rewritten and the error does not depend on future tokens, so  $\forall t : \|\mathbf{X}_t - \hat{\mathbf{X}}_t\|_\infty \leq B(1+\eta)$  holds sequence-wide under arbitrary multi-step decoding — about the actual (live) trajectory’s writes only.*

*Proof.* (i) The maximum over a union of sets is the maximum of the set maxima, each certified by Lemma 1. (ii) Immediate from the definition, since chunks are immutable after the write.

**Gate rejection under injected faults.** We inject three faults into the write path of Sec. 3.2 on a 512-token chunk and run the commit gates on the reconstruction from the *stored byte stream*: a wrong INT4 scale constant ( $s = m/6.5$ ), a mode map stale by one block, and a payload truncated by 5%. At  $B=0.2$ , gate G2 rejects all three — max abs err 0.225, 53.2, 4.23 against the  $B(1+\eta)$  threshold — the scale drift being the instructive case, a  $1.13\times$  violation only a check on the actual reconstruction can see. At  $B=0.01$  (the FP16-bypass regime), gate G3 additionally rejects the mode-map and truncation faults.

### 3.4 Transport Crossover Analysis

Under (1), the quantized path beats fp16 transport iff  $\alpha_{\text{codec}} + \beta_{\text{link}}(S_{16}/r) + T_{\text{consumer}}^q < \alpha_{16} + \beta_{\text{link}} S_{16} + T_{\text{consumer}}^{16}$ , where  $r$  is the *ledger-accurate* compression ratio: it counts every byte of the stored representation — payload, fp32 per-block scale bytes ( $S_{\text{meta}} = 4n_{\text{el}}/G = 2S_{16}/G$ ), and, for the adaptive codec, one mode byte per block ( $S_{16}$  is the uncompressed FP16 chunk bytes throughout). For a payload ratio  $q$ ,  $r = 1/(1/q + 2/G)$ , i.e., 1.939 for INT8 ( $q=2$ ) and 3.765 for INT4 ( $q=4$ ) at  $G=128$ . The byte-only (“free-lunch”) model sets  $\alpha_{\text{codec}} = \alpha_{16}$  and  $T_{\text{consumer}}^q = T_{\text{consumer}}^{16}$ ; equating both sides yields the closed-form crossover

$$S_{16}^* = \frac{\Delta\alpha + \Delta T_c}{\beta_{\text{link}}(1 - 1/r)}, \quad (3)$$

where  $\Delta\alpha \equiv \alpha_{\text{codec}} - \alpha_{16}$  is the fixed codec tax and  $\Delta T_c \equiv T_{\text{consumer}}^q - T_{\text{consumer}}^{16}$  is the consumer difference: the numerator is the fixed tax, so the true crossover sits at larger sizes than the byte-only prediction. For the *certified* path,  $\alpha_{\text{codec}}$  additionally contains the error pre-pass and the mode-selection kernel (Sec. 4.5 measures this tax). The crossover surface is analyzed on a pre-registered 192-cell protocol grid in Sec. 4.6.

### 3.5 Deployment Resolver: Measured Profiles to Operating Point

An operator hands the system a constraint triple and receives an operating point. We implement the *deployment resolver* in pure Python (no GPU): its

inputs are three *heterogeneous measured predicates* — a quality curve from model experiments, an interpolated transport grid from kernel measurements, a byte ledger from counting — and its output is their intersection, a single deployment verdict. The resolver is *not an online learning controller*; it consumes measured deployment profiles and current constraints, and the resolution is constraint filtering ( $\mathcal{Q} \cap \mathcal{L} \cap \mathcal{M}$ ), not online optimization.

The resolver maps the constraint triple  $(\varepsilon, \tau, M)$  (quality, latency, memory) to an operating point  $(B^*, \text{mix}(B^*), p^*, r(B^*))$  by constraint filtering: each predicate defines a feasible set over the measured candidate space — the budget candidates  $B$  of the per-model ratio curve (Sec. 4.2), the transport paths of the grid (PCIe and the measured same-GPU device-to-device-proxy legs, dequantize-then-use consumers), and the fp16 baseline — and the operating point is the highest-compression point of the intersection. The quality predicate is the *measured* perplexity degradation per model (Sec. 4.3), not a fitted proxy; the quality SLO levels used in the main text are strict ( $\Delta\text{PPL} \leq 1\%$ ), moderate ( $\leq 3\%$ ), and relaxed ( $\leq 5\%$ ). Feasibility uses only measured quantities:  $\mathcal{Q} = \{B : \Delta\text{PPL}^{\text{meas}}(B) \leq \varepsilon\}$ ;  $\mathcal{L} = \{(B, p) : T_{\text{path}}(B, S_{16}, p) \leq \tau\}$ , with  $T_{\text{path}}$  the measured grid cell interpolated to the transported size  $S_{16}/r(B)$  plus the certificate tax of Sec. 4.5; and  $\mathcal{M} = \{B : \text{bytes}(B, S_{16}) \leq M\}$  under the ledger. The resolution is: filter the measured candidate space by the three predicates, then select the highest-compression point of the intersection; if the intersection is empty, return INFEASIBLE with the lowest-latency candidate. The codec’s internals never enter the procedure, only the three measured predicates do.

The fp16 baseline is always a candidate, so the resolver answers “compression never pays here” (the same-GPU proxy verdict of Sec. 4.6) and “infeasible” (no measured point meets  $\tau$  or  $M$ ) as first-class outputs. The selection step is a policy, not a measurement: an operator who prefers latency over ratio swaps the argmax for an argmin over  $T_{\text{path}}$ . Uniform codecs appear as a-posteriori alternatives: their measured  $\Delta\text{PPL}$  is reported and enforced, but they carry no per-write bound.

### 3.6 Codec-Agnostic Resolution

At the interface boundary, *any* bounded-error KV representation is reduced to three measured curves — an *error curve* (knob  $\rightarrow$  reconstruction error /  $\Delta\text{PPL}$ ), a *latency curve* (knob, chunk size, link), and a *bytes curve* (knob  $\rightarrow$  ledger-accurate ratio) — and the resolution procedure above consumes exactly these three curves and nothing else: an external uniform-INT8 adapter (one fixed operating point, no pre-pass, no gate) resolves to a valid operating point in every scenario tested.

**A third-party codec.** We compile PackKV [6] (IPDPS 2026) from source and measure its released compressor (median of 50, RTX 4070 Ti SUPER): its construction error (0.64–0.85) excludes it from every tight-quality scenario and its measured round trip (2.9–3.7 ms per 1 MB) exceeds every latency SLO, so the verdict is fp16 bypass or explicit infeasibility; under the SLOs its curves can meet ( $\varepsilon=0.50$ ,  $\tau=3.5$  ms, 1 MiB) it resolves to its own feasible point (2.55 $\times$ , 3.0 ms). The attention-weighted repacking stage is not exercised; as a lossless

**Table 1.** Real-KV adaptive curves (protocol corpus; BLK=128 flat blocks; K/V mode shares INT4/INT8/FP16 of their block counts; “randn”: same-size Gaussian reference; G2 max err =  $B$  on every row). The curves diverge above  $B \approx 0.2$ : K blocks are INT8-bound.

| Model (ctx)       | $B$ | ratio | K 4/8/16    | V 4/8/16     | randn |                 |
|-------------------|-----|-------|-------------|--------------|-------|-----------------|
| TinyLlama (2048)  | 0.2 | 2.55  | .03/.97/.00 | .99/.01/.00  | 2.57  | Full six-budget |
| TinyLlama (2048)  | 0.5 | 2.61  | .09/.91/.00 | 1.00/.00/.00 | 3.71  |                 |
| Qwen2.5-3B (256)  | 0.2 | 2.20  | .00/.97/.03 | .58/.42/.00  | 2.57  |                 |
| Qwen2.5-3B (2048) | 0.2 | 2.16  | .00/.97/.03 | .52/.48/.00  | 2.57  |                 |
| Qwen2.5-3B (2048) | 0.5 | 2.66  | .18/.82/.00 | .96/.04/.00  | 3.71  |                 |

K/V counts are in the supplementary material.

permutation, the measured error is exact and the measured bytes bound the full pipeline’s.

## 4 Evaluation

Setup: NVIDIA RTX 4070 Ti SUPER (16 GB), CUDA 13.0, torch 2.13.0+cu130, all kernels Triton. Error and ratio numbers are kernel-deterministic (bit-exact arithmetic, fixed seeds); absolute timings follow the measurement rule of Sec. 3.1; every adaptive cell below carries the certificate (gate G2 passed on every reported cell).

### 4.1 Codec Characterization

The certified codec realizes every budget with the bound honored on the actual reconstruction (1 MB randn chunk,  $d_{kv}=2048$ , BLK=128):  $1.142\times$  at  $B=0.01$ ,  $1.925\times$  at  $B=0.03$  (all INT8; construction error 0.0201 — uniform INT8’s point, now *verified per write*),  $2.589\times$  at  $B=0.2$ , and  $3.710\times$  at  $B=0.5$  (all INT4); G2’s max abs err equals  $B$  on every row, and the uniform rows carry no per-write bound. Ratios are size-invariant; the deployment quantity is the *real-KV* curve of Sec. 4.2.

### 4.2 Real-KV Curves for the Resolver

The E1 curve is randn; deployment KV is not. Table 1 measures the same adaptive selection on the real KV of TinyLlama-1.1B and Qwen2.5-3B (protocol corpus, 256- and 2048-token contexts, K and V counted separately, flat 128-element blocks, G2 verified).

Two regimes separate the real curve from the randn curve. *Tight budgets* ( $B \leq 0.03$ ): *real KV compresses better than randn*. At  $B=0.01$  the real curves give  $1.32\times/1.10\times$  (TinyLlama/Qwen) versus the randn  $1.11\times$ : real V blocks are small-magnitude and pass INT8 even at  $B=0.01$ . *Loose budgets* ( $B \geq 0.2$ ): *the real curve flattens far below randn*. K magnitudes are  $m \approx 5\text{--}20$  at 1B and up to  $\sim 90$  at 3B (Sec. 4.4), so  $m/14$  exceeds  $B$  for most K blocks: at  $B=0.5$  only 9% (TinyLlama) and 18% (Qwen2.5-3B) of K blocks admit INT4, K stays

**Table 2.** Model calibration on the WikiText-2 standard benchmark (test split, fixed 512-token windows, L40), shift-by-one perplexity with KV written quantized at every layer;  $\Delta\text{PPL}$  is  $\text{ppl}_{\text{cell}}/\text{ppl}_{\text{base}} - 1$ . Adaptive rows carry the certified invariant (G2 max error =  $B$  exactly); uniform rows are a-posteriori reference points.

| Model          | Codec             | ratio | $\Delta\text{PPL}$ | max err |
|----------------|-------------------|-------|--------------------|---------|
| TinyLlama-1.1B | INT8, $g=128$     | 1.939 | +0.02%             | 0.109   |
| TinyLlama-1.1B | INT4, $g=128$     | 3.765 | +9.41%             | 1.939   |
| TinyLlama-1.1B | adaptive, $B=0.2$ | 2.549 | +0.50%             | 0.200   |
| TinyLlama-1.1B | adaptive, $B=0.5$ | 2.604 | +0.57%             | 0.500   |
| OPT-125M       | INT8, $g=128$     | 1.939 | +0.00%             | 0.047   |
| OPT-125M       | INT4, $g=128$     | 3.765 | +2.20%             | 0.839   |
| OPT-125M       | adaptive, $B=0.2$ | 2.563 | +0.70%             | 0.200   |
| OPT-125M       | adaptive, $B=0.5$ | 3.675 | +2.15%             | 0.500   |
| Qwen2.5-3B     | INT8, $g=128$     | 1.939 | +0.10%             | 0.380   |
| Qwen2.5-3B     | INT4, $g=128$     | 3.765 | +12462×            | 6.750   |
| Qwen2.5-3B     | adaptive, $B=0.2$ | 2.195 | +0.82%             | 0.200   |
| Qwen2.5-3B     | adaptive, $B=0.5$ | 2.742 | +2.46%             | 0.500   |

INT8, and the ratio saturates at  $2.61\times / 2.66\times$  — the randn curve’s  $3.71\times$  is *not achievable on real KV*. The curves reproduce the WikiText-2 adaptive rows of Table 2 (TinyLlama 2.549/2.604, Qwen 2.195/2.742): the same selection rule on two corpora; a ratio predicate instantiated on randn would over-predict the operating points, so the resolver uses the real curves.

### 4.3 Quality SLO on the Standard Benchmark

Table 2 is the SLO-constraining evidence on a held-out standard corpus: larger measured KV magnitudes coincide with larger reconstruction errors and more severe degradation. OPT-125M barely registers KV error (+2.20% at uniform INT4); TinyLlama-1.1B degrades measurably (+9.41%, outside even a 5% SLO at 1B scale); Qwen2.5-3B — the first model whose KV magnitudes push the uniform INT4 error to 6.75 — *collapses* ( $\Delta\text{PPL} \approx 12462\times$ ). The INT8 operating point is PPL-neutral on all three models (+0.02%, +0.00%, +0.10%), and the *certified adaptive* operating points satisfy a strict 1% PPL-degradation SLO at  $B=0.2$  ( $\Delta\text{PPL} \leq 0.82\%$  across the three models) and a moderate 3% SLO at  $B=0.5$  ( $\leq 2.46\%$ ). The uniform-INT4 collapse boundary is scale-dependent — at 3B scale the SLO-valid ratio collapses to the INT8  $1.94\times$  — and the certificate *detects* this: the error 6.75 is exactly what G2 verifies against  $B$  at write time, so a deployment choosing  $B=0.2$  is refused INT4 blocks whose error exceeds the budget.

### 4.4 Model- and Magnitude-Dependent KV Sensitivity (Mechanism)

The pattern is explained by the magnitude mechanism: the per-block max-scale error  $e_4 = m/14$  is proportional to the block magnitude  $m$ , and the worst-case per-block magnitude recovered from the *measured* reconstruction error grows  $9.8 \rightarrow 21.9 \rightarrow 92.8 \rightarrow 419$  across OPT-125M, TinyLlama-1.1B, Qwen2.5-3B, and

Qwen2.5-7B (cross-check: measured INT8 errors match  $m^*/254$  on every model). The growth is concentrated in the K path: per-block K magnitudes reach 92.8 on Qwen2.5-3B versus 20.9 on TinyLlama-1.1B, while V magnitudes stay below 28 on both (full probe and per-layer tables, and a simulated layer-sensitivity analysis, are in the supplementary material). *Larger observed K magnitudes make uniform INT4 infeasible on Qwen3B; the adaptive fallback recovers a valid operating point* — the certified  $B=0.2$  row of Table 2 stays within the 1% SLO at  $2.195\times$ , whereas the uniform INT4 cell collapses.

As a mechanism diagnostic, a random multi-head attention proxy with K and V compressed at budget  $B$  fits  $e_{\text{out}}(B) = 0.668B + 0.0013$  ( $R^2 = 0.856$ ) — a nearly linear amplification of KV error through  $\text{softmax}(QK^\top)V$  (full table in the supplementary material). The resolver does *not* use this law: the per-model PPL table of Sec. 4.3 is the quality predicate, and the law over-predicts the SLO-satisfying budget at  $B=0.3$  (measured  $0.28 > 0.25$ ).

#### 4.5 Codec Timing and the Certificate Tax

The certified path’s cost is real and reportable: at 1 MB and  $B=0.3$ , the pre-pass plus mode selection are  $27.6\mu\text{s}$  — 62% of the adaptive path and  $2.2\times$  the uniform-INT8 codec time (per-kernel medians in the supplementary material; co-tenant-load flag; intra-machine comparisons only). This “certificate tax” belongs in the transport ledger: it shifts  $\alpha_{\text{codec}}$  and therefore the crossover  $S_{16}^*$  (3) to the right. The same kernels, measured end to end in the two-L40 pipeline of Sec. 5.3, scale linearly in the chunk (pre-pass 0.14 ms + selection 0.10 ms + pack 0.14 ms at 46 MB).

#### 4.6 Transport Crossover: The Full Grid

The latency half of the paper is governed by the pre-registered 192-cell protocol grid, measured on an idle NVIDIA A100-SXM4-40GB (12 sizes  $\times$  2 transfer modes  $\times$  codec/group configs  $\times$  2 consumers, 64 KB–16 MB). Two patterns stand out. First, *the transport link decides whether compression ever pays*: every measured same-GPU device-to-device-proxy path has  $S_{16}^* = \infty$  — the codec’s fixed cost ( $\alpha_{\text{codec}} \approx 53\text{--}92\mu\text{s}$ ) is never repaid on the overhead-dominated proxy (no measured true-NVLink crossover is claimed). On the PCIe leg the byte cost is  $\beta_{\text{link}}=42.6\text{ ps/B}$  (23.5 GB/s marginal, consistent with PCIe Gen4), and the dequantize-then-use paths cross over at 1.06–2.08 MB (int4/{128, 64}: 1056/1398 KB; int8/{128, 64}: 1435/2081 KB); a byte-only model would predict the same crossover on both links. Second, *the INT8-GEMM consumer never pays*: the “consum” paths have  $S_{16}^* = \infty$  on both links, because the dequantization tax  $dT_c$  (1355–2766  $\mu\text{s}$  in the scale-GEMM leg) dominates the transport savings in every cell up to 16 MB. The certified path pays the pre-pass tax on top, so its crossover shifts further right; the grid is measured for *serialized* codec–transfer–decode execution.

#### 4.7 Resolver Decisions on Real-Model Profiles

Table 3 instantiates the resolver (Sec. 3.5) with the measured profiles: the quality predicate is the per-model WikiText-2 table of Sec. 4.3, the ratio predicate is

**Table 3.** Deployment-resolver operating points on real-model profiles (measured constants from Secs. 4.3, 4.2, 4.5 and the 192-cell grid).  $S_{16}$ =2 MiB fp16 chunk, PCIe unless stated; “Bytes”: ledger-accurate transported bytes  $S_{16}/r$ ;  $T_{\text{path}}$ : measured grid cell at that byte count + certificate tax (27.6  $\mu$ s) for certified rows; parenthetical: difference versus the fp16 path (194  $\mu$ s on PCIe, 55  $\mu$ s on the proxy). Certified rows carry gate G2; “a-posteriori” rows do not.

| Model, SLO                            | Resolver point            | ratio | $\Delta$ PPL | Bytes        | $T_{\text{path}}$ ( $\mu$ s) |
|---------------------------------------|---------------------------|-------|--------------|--------------|------------------------------|
| TinyLlama-1.1B, $\leq 1\%$            | certified $B=0.2$         | 2.549 | +0.50%       | 804 KB       | 160 (−34)                    |
| TinyLlama-1.1B, $\leq 3\%$            | certified $B=0.5$         | 2.604 | +0.57%       | 787 KB       | 160 (−34)                    |
| OPT-125M, $\leq 1\%$                  | certified $B=0.2$         | 2.563 | +0.70%       | 799 KB       | 160 (−34)                    |
| OPT-125M, $\leq 3\%$                  | certified $B=0.5$         | 3.675 | +2.15%       | 557 KB       | 158 (−35)                    |
| OPT-125M, $\leq 5\%$                  | uniform INT4<br>(a-post.) | 3.765 | +2.20%       | 544 KB       | 130 (−64)                    |
| Qwen2.5-3B, $\leq 1\%$                | certified $B=0.2$         | 2.195 | +0.82%       | 933 KB       | 159 (−34)                    |
| Qwen2.5-3B, $\leq 3\%$                | certified $B=0.5$         | 2.742 | +2.46%       | 747 KB       | 160 (−34)                    |
| any model, —                          | fp16 (same-GPU proxy)     | 1.00  | —            | 2 MiB        | 55                           |
| TinyLlama-1.1B, $\leq 3\%$ $M=700$ KB | <i>infeasible</i>         | —     | —            | 787 KB > $M$ | —                            |

Under  $\leq 5\%$ , the TinyLlama and Qwen rows coincide with the  $\leq 3\%$  rows (their uniform-INT4 cells violate the SLO: +9.41%, +12462 $\times$ ); OPT’s INT4 cell (+2.20%) is admitted. All numbers are recomputed from the JSON the tables are drawn from (prototype/sla.controller.py).

the per-model real-KV curve (the WikiText-2 ratios of Table 2), and the latency predicate is the 192-cell grid interpolated at the transported byte count plus the certificate tax. The candidate set is {FP16, INT8,  $B=0.2$ ,  $B=0.5$ , INT4}, and quality SLOs map to budgets per model:  $\leq 1\% \rightarrow B=0.2$  (Qwen measures +0.82%),  $\leq 3\% \rightarrow B=0.5$  (Qwen +2.46%), and the relaxed  $\leq 5\%$  SLO exposes the uniform-INT4 boundary.

Three conclusions follow. First, *the same SLO admits different points per model*: under  $\leq 5\%$ , the resolver chooses uniform INT4 (3.765 $\times$ , no per-write bound) for OPT-125M but stays at the certified  $B=0.5$  point for TinyLlama-1.1B and Qwen2.5-3B, whose INT4 cells violate the SLO. Second, *the link decides the transport verdict*: on the same-GPU proxy every compressed path has  $S_{16}^* = \infty$ , so the resolver returns fp16 at 55  $\mu$ s — compression never pays there — while on PCIe the certified points beat fp16 transport by 34–35  $\mu$ s at 2 MiB, and uniform codecs, free of the certificate tax, win by more. Third, *infeasibility is explicit*: with a 700 KB memory budget the TinyLlama  $\leq 3\%$  row resolves to *infeasible* (the certified  $B=0.5$  point needs 787 KB), and a latency SLO below the fastest measured path (130  $\mu$ s) rejects every candidate. The summary object is the feasible region  $\mathcal{F}(S_{16}, L)$ : non-empty on PCIe above 1.06–2.08 MB for the certified dequantize-then-use paths and empty on the same-GPU proxy. The two-worker demo of Sec. 5.3 executes one such point end to end.

## 5 Serving Integration

A deployment decision that cannot plug into a production serving stack is a curiosity; one that plugs in should do so through the stack’s *existing abstractions*, not by forking them. vLLM’s paged-attention path exposes exactly the three touch points the framework needs: the KV write hook, a first-class certificate object, and the compressed transport path of the disaggregated-prefill machinery.

### 5.1 Integration points

(i) **The KV write hook.** vLLM’s attention worker writes K/V into the paged arena via `reshape_and_cache_flash`; the hook sits between the projection outputs and that op: per 128-element block it computes the candidate INT4/INT8 errors (pre-pass), selects the mode under the budget  $B$  (2), packs payload plus per-block fp32 scales and mode bytes, and verifies gate G2 on the actual reconstruction before the write commits (fp16 fallback on violation).

(ii) **The certificate object.** The per-write outcome is a first-class value an operator can inspect and log: `Certificate{B, max_abs_err, modes, verified, layer, step}`, compositional per Lemma 2. In disaggregated deployments the certificate travels with the chunk: the envelope is the packed KV *plus* its mode map, scales, the budget  $B$ , the writer’s G2 verdict, and a payload digest, so the decode worker can verify transfer integrity and trust the bound without re-running the codec.

(iii) **The compressed transport path.** The export op of vLLM’s disaggregated-prefill machinery extracts per-token KV through the slot mapping; the compressed transport replaces the fp16 payload with (packed payload, scales, mode bytes) — exactly the byte accounting of Sec. 3.1 — and the import side writes through `reshape_and_cache_flash`. The transport cost model of Sec. 3.4 applies unchanged.

### 5.2 Certificate on live paged KV

The prototype runs the (i)+(ii) logic on the *live* paged KV of a running vLLM 0.26 engine: one 128-token prefill of TinyLlama-1.1B (22 layers, 4 KV heads, 64-dim head, 16-token blocks, 2.75 MiB of fp16 KV), then, for every layer and both K and V, the adaptive codec and gate G2 at budgets  $B \in \{0.01, 0.1, 0.2, 0.3, 0.5\}$ . G2 passed on every layer, both tensors, all five budgets — 220 verified writes; at  $B=0.01$ , 599 INT4 / 5167 INT8 / 5498 FP16 blocks (real KV needs more FP16 bypass than the randn cells of Sec. 4.1), and at  $B=0.5$ , 6139 INT4 / 5125 INT8 / 0 FP16. The write path is closed: the quantized payload was scattered back into the live paged arena and re-extracted, matching the certificate’s reconstruction (max abs deviation  $0.5 = B$  at the  $B=0.5$  cell), with block bounds composing per Lemma 2 to the per-token rows the attention GEMM consumes. Codec cost at serving granularity: on one layer’s K tensor (65 KB fp16), the uniform INT4 Triton codec costs 0.058 ms; the fused codec of Sec. 5.3 costs 0.57–1.16 ms on whole 23–92 MB chunks.

**Table 4.** Two-worker semantic and transport-phase validation, TinyLlama-1.1B, 256-token prefill, 32-token decode, single A100, same-host two-process PCIe proxy (median of 25 repetitions). Bytes are ledger-accurate; transfer = D2H + host relay + H2D medians; G2: write-time gate verdict; match: greedy continuation vs. the fp16 arm (32/32 for all arms, and vs. the no-transport local reference).

| Arm                | Bytes (KB) | Transfer ( $\mu$ s) | G2               | Token match |
|--------------------|------------|---------------------|------------------|-------------|
| fp16 (no codec)    | 5632       | 832                 | n/a (bit-exact)  | 32/32       |
| certified, $B=0.2$ | 2205       | 402                 | pass (err 0.199) | 32/32       |
| uniform INT8       | 2904       | 479                 | n/a              | 32/32       |
| uniform INT4       | 1496       | 308                 | n/a              | 32/32       |

### 5.3 Two-worker semantic and transport-phase validation

To close the loop end to end, prefill worker A and decode worker B are two OS processes on one A100, and the “network” between them is a CPU-side relay (a /dev/shm file handshake): every KV byte leaves the GPU (D2H), crosses host memory, and re-enters the GPU (H2D) — a faithful proxy for the PCIe host-transfer leg of the ledger. Worker A is a real vLLM 0.26 engine (TinyLlama-1.1B, fp16): it prefills a 256-token prompt, extracts the live paged KV through the official export path, and, per arm, runs the codec and packs the certificate envelope. Worker B is a transformers decode worker on the same model: it reads the envelope, dequantizes into a `DynamicCache`, and decodes 32 tokens greedily — computing *no* prefill attention. Four arms are measured (fp16; certified adaptive  $B=0.2$ ; uniform INT8; uniform INT4), each over 25 repetitions with per-repetition CUDA-event timing and a sha256 check at the receiver (Table 4).

The current reference integration uses an unoptimized Python adaptive packing path (4.0s for the certified arm); we therefore use this experiment only to validate semantics and transport-phase accounting, while performance conclusions come from the separate Triton kernel grid. Three findings follow. First, *the certificate object crosses the relay intact*: the sha256 digest check passed at the receiver on every arm, and the writer’s G2 verdict held on the actual packed representation (max abs err  $0.199 \leq B(1+10^{-3})$  at  $B=0.2$ ); a per-layer cross-check of the fp16 arm against worker B’s local prefill KV showed a max absolute difference of 0.039 — fp16 rounding level. Second, *the byte ledger transfers to a real pipeline*: transported bytes drop from 5632 KB (fp16) to 2205 KB (2.55 $\times$ , certified adaptive) and 1496 KB (3.77 $\times$ , uniform INT4), and for this 5.5 MiB chunk — above the measured 1.06–2.08 MB crossover — transfer latency falls from 832 to 402  $\mu$ s (certified adaptive) and 308  $\mu$ s (uniform INT4), consistent with the grid of Sec. 4.6. Third, *decode quality and throughput are measured per arm*: decode is flat at 53–55 tok/s, and every arm reproduced the fp16-arm continuation exactly — the quality claim everywhere is the verified bound (G2), not the match rate.

*Fused Triton codec, real dual-GPU PCIe P2P (L40s).* The two-L40 pipeline runs the *fused* codec of Sec. 4.5 end to end: worker A prefills (512–4096 tokens), the fused kernels (error pre-pass  $\rightarrow$  mode selection under  $B \rightarrow$  pack; one Triton kernel per stage) write the certificate envelope, the envelope crosses *device-to-device* via PCIe P2P (`tensor.to`, no host relay), and worker B’s fused

**Table 5.** Fused-codec end-to-end, two L40s, real PCIe P2P, TinyLlama-1.1B, 128-token decode. codec: fused Triton pre-pass+selection+pack; xfer: P2P envelope transfer (payload + fp32 scales + mode bytes); e2e: prefill + codec + xfer + unpack + decode. G2 max err =  $B$  on every certified row; sha256 verified at the receiver; decode is flat across arms (36–60 tok/s).

| Prefill<br>(tokens) | Arm               | codec<br>(ms) | xfer<br>(ms) | codec+xfer<br>(ms) | e2e<br>(ms) |
|---------------------|-------------------|---------------|--------------|--------------------|-------------|
| 1024                | fp16              | —             | 0.89         | 0.89               | 2481        |
| 1024                | certified $B=0.2$ | 0.57          | 0.39         | 0.96               | 2475        |
| 2048                | fp16              | —             | 1.76         | 1.76               | 2533        |
| 2048                | certified $B=0.2$ | 0.69          | 0.73         | 1.42               | 2564        |
| 2048                | certified $B=0.5$ | 0.68          | 0.72         | 1.40               | 2712        |
| 4096                | fp16              | —             | 3.51         | 3.51               | 3650        |
| 4096                | certified $B=0.2$ | 1.02          | 1.42         | 2.44               | 3046        |
| 4096                | certified $B=0.5$ | 1.16          | 1.38         | 2.54               | 3231        |

Ratios  $2.55 \times (B=0.2)$

and  $2.61\text{--}2.62 \times (B=0.5)$  on the 2048-token cell, matching the real-KV curve of Sec. 4.2 block-for-block. 128-token continuation match vs. the fp16 arm: 0.84 (512-token cell), 0.96–0.97 (1024–4096).

unpack/dequantize kernel restores the cache before a 128-token greedy decode with zero prefill attention (Table 5).

Three findings replace the previous paragraph’s Python-bound figures. First, *the codec is no longer the bottleneck*: the fused codec costs 0.57–1.16 ms for 23–92 MB of fp16 KV (per-kernel medians at 46 MB: pre-pass 0.14 ms, selection 0.10 ms, pack 0.14 ms), 39% of the fp16 transfer there. Second, *the codec+transfer crossover is now measured in the real pipeline*: the certified path’s codec+transfer crosses under the fp16 transfer between 23 and 46 MB of fp16 KV (at 11.5 MB it is still  $2.1 \times$ ; at 92 MB 2.44 vs. 3.51 ms) — the same-host grid’s 1.06–2.08 MB crossover does not transfer to the real 26 GB/s P2P link, which is  $\sim 1000 \times$  faster than the proxy’s host round trip: the grid’s *existence*, not its absolute size, is link-independent. Third, *the byte ledger and the certificate hold on the real link*: ratios and mode mixes match the real-KV curves (Sec. 4.2) block-for-block, G2 max error =  $B$  exactly on every certified arm, the sha256 digest verifies at the receiver, and the 128-token continuation matches the fp16 arm at 0.84–0.97. End-to-end latency is decode-dominated and flat across arms (2.5–4.0 s); the transport savings land in the transfer phase, where the ledger puts them.

## 5.4 Scope

The proxy demo’s two workers share one GPU and a /dev/shm relay instead of a NIC (the L40 P2P validation removes the relay but not the shared process host), worker B consumes the KV through a reference transformers path rather than vLLM’s own attention kernels. The measured claims are narrower: *the certified codec executes on live vLLM paged KV with its per-write cost included in the ledger, the certificate object survives an end-to-end two-worker transfer with integrity and quality verified — over the same-host proxy and over a real dual-GPU PCIe P2P link — and the byte ledger of Sec. 3.4 predicts the transfer phase of the pipeline.*

## 6 Limitations

**Randn vs. real KV.** The E1 codec curve and the attention-error diagnostic use Gaussian values; the resolver’s ratio and quality predicates use the real-KV curves (Sec. 4.2) and the per-model WikiText-2 table (Sec. 4.3). The certificate is distribution-agnostic by construction; the ratio–error curve is not.

**Corpus and SLO looseness.** The 1%/3%/5% PPL-degradation SLO rows are calibrated on WikiText-2 across three architectures; TinyLlama-1.1B registers +9.4% at uniform INT4, outside the 5% SLO at 1B scale. The 50% SLO appears in the supplementary material only, delimiting the uniform-INT4 scale boundary.

**Timing provenance.** The codec timing ledger (Sec. 4.5, RTX 4070 Ti SUPER) was taken under a co-tenant-load flag and only intra-machine comparisons are made from it; the 192-cell transport grid (Sec. 4.6) is on an idle A100 and is clean. Energy-aware deployment is orthogonal to the latency–quality–memory contract studied here and is left for future work.

**Codec and testbed scope.** INT2/INT3 and per-layer budgets (distinct  $B_l$ ) were not implemented; the measured same-GPU proxy is a same-GPU copy, not NVLink; the INT8 consumer is a sequence of small `torch._int_mm` calls (conservative for the latency half); and executions are serialized (overlap is outside the ledger). The two-worker demo (Sec. 5.3) shares one GPU and a `/dev/shm` relay instead of a NIC, and the decoding side is a reference transformers path, not vLLM’s own attention kernels. E1–E4 are on one RTX 4070 Ti SUPER; the transport grid and the vLLM prototype on one A100; the two-worker validation and the fused codec on two idle L40s.

## 7 Conclusion

We studied when KV cache compression is actually beneficial in cloud LLM serving. The key result is that byte reduction alone is insufficient: codec and verification overhead create a link- and size-dependent crossover, while model-dependent KV magnitudes constrain the admissible compression ratio. By combining measured quality, transport, and memory profiles, the deployment resolver selects a feasible operating point or explicitly rejects compression: across our measured PCIe path, compressed dequantize-then-use paths become beneficial above approximately 1.06–2.08 MB, whereas the same-GPU proxy favors FP16. Real-model and live-KV experiments further show why a fixed compression ratio is not universally deployable. These results argue for treating KV compression as a deployment decision rather than an unconditional serving optimization.

## References

1. Ainsworth, M., Tugluk, O., Whitney, B., Klasky, S.: Multilevel techniques for compression and reduction of scientific data—quantitative control of accuracy in derived quantities. *SIAM Journal on Scientific Computing* **41**(4), A2146–A2171 (2019). <https://doi.org/10.1137/18M1208886>

2. Calver, D.: Runtime-certified bounded-error quantized attention (2026). <https://doi.org/10.48550/ARXIV.2605.20868>
3. Di, S., Cappello, F.: Fast error-bounded lossy HPC data compression with efficient synchronous multi-phase streaming and multi-level adaptation. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2016). <https://doi.org/10.1109/IPDPS.2016.11>
4. Guo, Y., Joshi, S.: SplitZip: Ultra fast lossless KV compression for disaggregated LLM serving (2026). <https://doi.org/10.48550/ARXIV.2605.01708>
5. Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M.W., Shao, Y.S., Keutzer, K., Gholami, A.: KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In: Advances in Neural Information Processing Systems (NeurIPS) (2024)
6. Jiang, B., Yang, T., Liu, Y., He, X., Di, S., Jin, S.: PackKV: Reducing KV cache memory footprint through LLM-aware lossy compression. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 598–611 (2026). <https://doi.org/10.1109/IPDPS65963.2026.00057>
7. Lab, M.: WitCert: Sound runtime risk observability and gating for KV-cache quantization (2026)
8. Lin, Y., Tang, H., Yang, S., Zhang, Z., Xiao, G., Gan, C., Han, S.: QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In: Machine Learning and Systems (MLSys) (2025)
9. Lindstrom, P.: Fixed-rate compressed floating-point arrays. IEEE Transactions on Visualization and Computer Graphics **20**(12), 2674–2683 (2014). <https://doi.org/10.1109/TVCG.2014.2346458>
10. Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., Maire, M., Hoffmann, H., Holtzman, A., Jiang, J.: CacheGen: KV cache compression and streaming for fast large language model serving. In: ACM SIGCOMM Conference (SIGCOMM). pp. 38–56 (2024). <https://doi.org/10.1145/3651890.3672274>
11. Liu, Z., Ma, X., Luo, D., Zhao, H., Lu, B., Huang, W., Gu, Y., Liu, X., Wei, Z., Tao, D., Tan, G., Liu, J.: KVServe: Service-aware KV cache compression for communication-efficient disaggregated LLM serving. In: ACM SIGCOMM Conference (SIGCOMM) (2026)
12. Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., Hu, X.: KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In: International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research, vol. 235, pp. 32332–32344. PMLR (2024)
13. Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., Bianchini, R.: Splitwise: Efficient generative LLM inference using phase splitting. In: IEEE/ACM International Symposium on Computer Architecture (ISCA) (2024). <https://doi.org/10.1109/ISCA59077.2024.00090>
14. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., Han, S.: SmoothQuant: Accurate and efficient post-training quantization for large language models. In: International Conference on Machine Learning (ICML). Proceedings of Machine Learning Research, vol. 202, pp. 38087–38099. PMLR (2023)
15. Zhang, Z., Shen, H., Vargaftik, S., Basat, R.B., Mitzenmacher, M., Yu, M.: HACK: Homomorphic acceleration via compression of the key-value cache for disaggregated LLM inference. In: ACM SIGCOMM Conference (SIGCOMM) (2025). <https://doi.org/10.1145/3718958.3750481>