KVShuttle: When and How to Compress KV Caches for Disaggregated LLM Serving

Awneesh Verma
awneeshtiwari@gmail.com

Abstract

Disaggregated prefill/decode serving separates LLM inference across GPUs, creating a KV cache transfer bottleneck between them. While numerous KV cache compression methods exist, they were designed for memory reduction—not transfer optimization—and have never been compared under transfer-realistic conditions. We present **KVShuttle**, the first systematic benchmark evaluating 14 compression strategies across 6 models, measuring the full compress–transfer–decompress pipeline with GPU-calibrated timing on Tesla T4 and A100 GPUs. We find that: (1) the optimal strategy depends strongly on network bandwidth and model architecture—no single method dominates; (2) without GPU-accelerated kernels, compression only helps below 1 Gbps; with GPU pipelining, the break-even extends to 50–100 Gbps; (3) cosine similarity is a poor predictor of generation quality—INT4 achieves 0.99 cosine similarity yet produces near-random output on some models; (4) model sensitivity varies dramatically, with Qwen models suffering catastrophic failure under INT4 while Llama/Mistral maintain >95% token agreement. Based on these findings, we provide a practitioner decision framework and release our benchmark toolkit at `https://github.com/awneesht/KVShuttle`.

1 Introduction

Disaggregated prefill/decode serving [1, 2] has become the standard architecture for production LLM inference, separating compute-intensive prefill from memory-bound decode onto specialized GPU pools. This separation introduces a new bottleneck: the KV cache computed during prefill must be transferred to the decode GPU before generation can begin.

For a Llama-3.1-8B model with a 2048-token prompt, the KV cache is 128 MB in FP16. At 10 Gbps Ethernet, this transfer takes over 100 ms—comparable to the prefill computation itself. As prompt lengths grow with long-context applications, KV cache transfer increasingly dominates end-to-end latency.

Numerous KV cache compression methods have been proposed [3, 4, 5], but they were designed to reduce *memory footprint* during inference, not to optimize *network transfer*. A method that halves KV cache size is only beneficial for transfer if its compression and decompression overhead is less than the transfer time saved—a condition that depends on network bandwidth, GPU compute speed, and whether compression can be pipelined with transfer.

No prior work systematically evaluates compression methods through this transfer lens. We address this gap with KVShuttle, contributing:

1. **A unified benchmark** of 14 compression strategies spanning quantization, low-rank projection, token pruning, and hybrid methods, evaluated across 6 models (3B–8B parameters) with over 7,000 experimental data points.

Preprint. Under review.

Table 1: Compression strategies evaluated in KVShuttle. Ratio, cosine similarity, and token agreement are means over 5 models and 50 WikiText prompts each (250 total evaluations per compressor). Δppl = perplexity delta relative to uncompressed.

Strategy	**Family**	**Ratio**	**Key cos**	**Token Agr.**	**Δppl**
identity	baseline	1.0×	1.0000	1.000	0.00
uniform_int8	quantization	2.0×	0.9998	0.972	0.03
cachegen	structured	3.5×	0.9914	0.925	0.81
kivi_2bit	quantization	6.6×	0.9465	0.737	7.50
palu_lr	low-rank	2.6×	0.9578	0.667	24.1
uniform_int4	quantization	3.6×	0.9877	0.593	13,294
cascade	prune+quant	7.1×	0.7212	0.332	17,982

2. **GPU-calibrated timing** with zero-copy CUDA Event measurement on Tesla T4 and A100 GPUs, showing 34–355× speedup over CPU baselines.
3. **End-to-end generation quality metrics** (perplexity delta and token agreement) that reveal cosine similarity's inadequacy as a quality proxy.
4. **A practitioner decision framework** with break-even bandwidth analysis and a compressor selection flowchart.

2 Background and Motivation

Disaggregated serving. In disaggregated LLM serving, prefill and decode run on separate GPUs [1, 2]. After prefill computes the KV cache for a prompt, the cache must be transmitted to a decode GPU via the datacenter network (typically 10–400 Gbps). The decode GPU then continues autoregressive generation using the received cache.

KV cache compression landscape. Existing methods span several families: *quantization* (uniform INT8/INT4 [11], KIVI [3], KVQuant [5]), *structured coding* (CacheGen [4], HACK [6]), *low-rank projection* (Palu [7]), and *token pruning* [12]. Each paper evaluates its method in isolation against 2–3 baselines, making cross-method comparison impossible.

The transfer gap. KVPress [8] benchmarks 25+ token eviction methods but only measures quality and memory—no transfer cost. The closest work is HACK [6], which accounts for compress/decompress overhead in its evaluation, but compares only 3 quantization methods. No existing work provides a cross-family benchmark under transfer-realistic conditions with GPU-calibrated timing.

3 KVShuttle Benchmark Design

3.1 Compression Strategies

We evaluate 14 strategies across four families (Table 1):

Quantization: Uniform per-tensor INT8 and INT4, FP8 (E4M3), KIVI 2-bit (per-channel keys, per-token values), KVQuant 2-bit (non-uniform with outlier handling). **Structured coding:** CacheGen (delta encoding with grouped quantization). **Low-rank:** Palu (SVD projection). **Token pruning:** Top-k 50% retention. **Hybrids:** Cascade (prune 50% then INT4), mixed precision (INT8 keys / INT4 values), Palu+INT4.

All compressors implement a common `BaseCompressor` interface operating on NumPy arrays of shape [layers, heads, seq, dim], ensuring fair comparison independent of framework optimizations.

3.2 Pipeline Model

We model the full transfer pipeline as:

$$T_{\text{total}} = T_{\text{compress}} + T_{\text{serialize}} + T_{\text{transfer}} + T_{\text{deserialize}} + T_{\text{decompress}} \quad (1)$$

Table 2: Token agreement by model and compressor. Qwen models suffer catastrophic failure under INT4 quantization while Llama and Mistral remain robust.

Model	**INT8**	**CacheGen**	**INT4**	**KIVI**	**Cascade**	**Palu**
Qwen2.5-3B	0.983	0.925	**0.087**	0.664	**0.070**	0.678
Phi-3.5-mini	0.965	0.926	0.903	0.730	0.618	0.463
Llama-3.1-8B	0.977	0.962	0.951	0.766	0.413	0.720
Mistral-7B	0.982	0.952	0.957	0.785	0.540	0.731
Qwen2.5-7B	0.954	0.861	**0.068**	0.739	**0.017**	0.741

where $T_{\text{transfer}} = \text{compressed\_size}/\text{bandwidth}$. Compression and transfer can be pipelined (overlapped), reducing total latency:

$$T_{\text{pipelined}} = T_{\text{compress}} + T_{\text{transfer}} + T_{\text{decompress}} - T_{\text{overlap}} \tag{2}$$

Compression and decompression times are measured directly (CPU NumPy baselines and GPU PyTorch kernels with CUDA Event timing). Transfer time uses an analytical model validated against real TCP localhost measurements across 1 KB–100 MB payloads.

3.3 Models and Evaluation

We evaluate on 6 instruction-tuned models spanning 3B–8B parameters: Qwen2.5-3B, Llama-3.2-3B, Phi-3.5-mini (3.8B), Qwen2.5-7B, Llama-3.1-8B, and Mistral-7B-v0.3. Models are loaded in FP16 via HuggingFace Transformers. KV caches are extracted from 50 WikiText-103 prompts (128–512 tokens).

Quality metrics. Beyond per-element cosine similarity and MSE, we measure *token agreement* (fraction of positions where greedy argmax matches the uncompressed baseline) and *perplexity delta* (change in cross-entropy loss). These require injecting the reconstructed KV cache into the model and running a forward pass on continuation tokens—an evaluation methodology absent from most prior work.

4 Results

4.1 Cosine Similarity Is Misleading

Our most striking finding is that cosine similarity—the standard quality metric in KV compression papers—poorly predicts generation quality. Uniform INT4 achieves 0.988 mean key cosine similarity, yet produces a perplexity delta of 13,294 and only 59.3% token agreement (Table 1). In contrast, CacheGen at slightly lower cosine similarity (0.991) achieves 92.5% token agreement and a perplexity delta of only 0.81.

The explanation is that small per-element quantization errors compound through multi-head attention, where keys and queries interact multiplicatively. High element-wise fidelity does not guarantee faithful attention patterns, especially when errors are correlated across heads or layers.

Takeaway: Papers evaluating KV compression via cosine similarity alone risk overstating quality preservation. End-to-end metrics like token agreement and perplexity delta are essential.

4.2 Model Sensitivity Varies Dramatically

Table 2 reveals stark model dependence. Qwen models collapse under INT4 quantization (6.8–8.7% token agreement, i.e., near-random), while Llama-3.1-8B and Mistral-7B maintain >95% agreement with the same compressor. This sensitivity is not visible in cosine similarity (Qwen INT4 cosine is still 0.98) and would be missed without end-to-end evaluation.

The pattern is consistent: models with fewer KV heads (Qwen uses 2–4 GQA heads) are more sensitive, likely because quantization errors in each head affect a larger fraction of the attention computation.

Table 3: GPU kernel speedup over CPU NumPy (mean across sequence lengths and model configs). Without GPU acceleration, compression only helps below 1 Gbps.

	Tesla T4		**A100-SXM4**	
Compressor	**Comp.**	**Decomp.**	**Comp.**	**Decomp.**
uniform_int8	34×	80×	190×	456×
kivi_2bit	53×	68×	284×	373×
uniform_int4	63×	62×	334×	311×
fp8_e4m3	36×	64×	355×	456×
cachegen	33×	56×	106×	275×
cascade	60×	48×	276×	223×
palu_lr	5×	216×	8×	848×

Table 4: Maximum network bandwidth at which compression reduces total transfer time. Above these thresholds, compression overhead exceeds transfer savings.

Strategy	**CPU Seq.**	**GPU Seq.**	**GPU Pipelined**
uniform_int8	≤1	≤25	≤50 Gbps
kivi_2bit	≤1	≤25	≤50 Gbps
uniform_int4	—	≤25	≤50 Gbps
fp8_e4m3	—	≤10	≤50 Gbps
cachegen	≤1	≤10	≤25 Gbps
cascade	—	≤50	≤100 Gbps
palu_lr	—	—	—

4.3 GPU Acceleration Is Essential

Table 3 shows that GPU kernels are 34–355× faster than CPU NumPy for compression/decompression. This is critical: with CPU-only compression, our model sweep (5,040 results across 6 models) shows that only 3 compressors achieve any speedup, and only at 1 Gbps—the slowest bandwidth tested. At datacenter speeds (≥10 Gbps), CPU compression overhead exceeds the transfer savings for every method.

Palu is an outlier: SVD-based compression achieves only 5–8× GPU speedup (matrix decomposition parallelizes poorly), but decompression (a matrix multiply) achieves 216–848×. This asymmetry makes Palu viable only in asymmetric deployment scenarios where the prefill GPU is fast but the network is very slow.

A100 speedups scale roughly linearly with memory bandwidth (A100/T4 bandwidth ratio is 6.4×; observed speedup ratio is 4–6×), suggesting our results generalize to other GPU tiers.

4.4 Break-Even Bandwidth Analysis

Table 4 shows the maximum bandwidth at which each strategy is beneficial. With GPU-pipelined execution, uniform INT8, KIVI, INT4, and FP8 break even at ∼50 Gbps—covering most Ethernet-based datacenter networks. Cascade (prune+quantize) extends to 100 Gbps due to its high compression ratio (7.1×), though at severe quality cost. Palu never achieves a net speedup due to its SVD compression bottleneck.

Practical implication: At NVLink speeds (≥400 Gbps), no compression is beneficial. For InfiniBand (100–200 Gbps), only cascade might help, but its quality loss is unacceptable for most applications. Compression is most valuable on Ethernet networks (10–50 Gbps), where uniform INT8 provides a 2× size reduction with 97% token agreement.

5 Practitioner Guidelines

Based on our analysis, we provide actionable recommendations (Figure 1):

1. **>50 Gbps bandwidth:** Do not compress. Transfer overhead dominates.

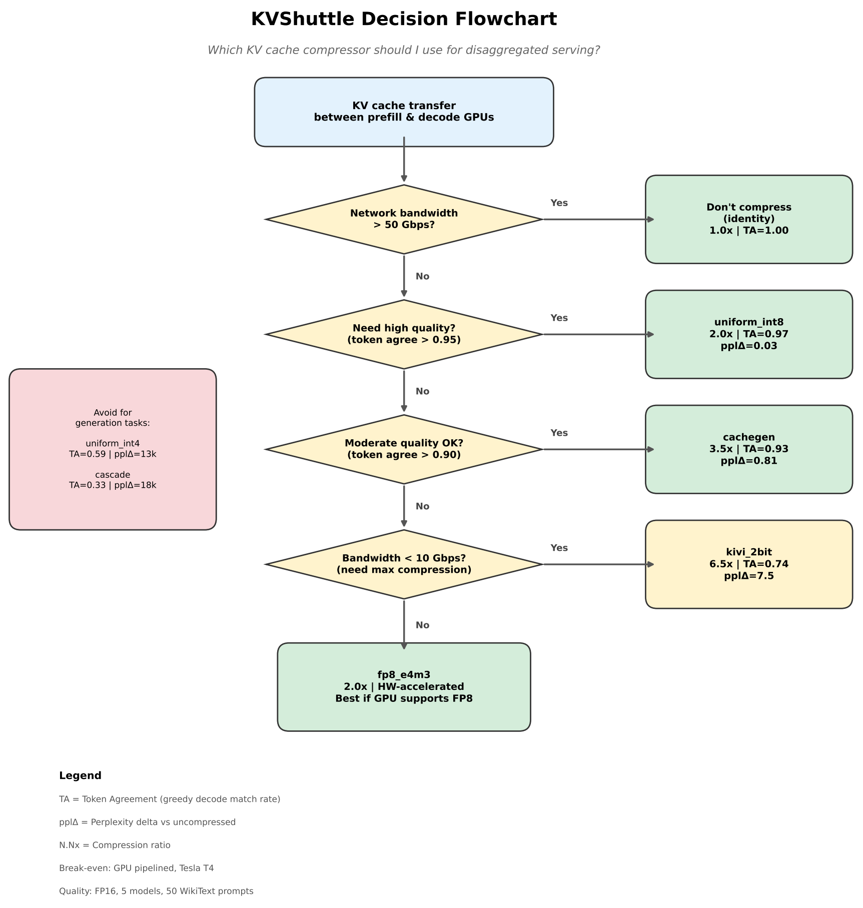
KVShuttle Decision Flowchart
Which KV cache compressor should I use for disaggregated serving?
KV cache transfer between prefill & decode GPUs
Network bandwidth > 50 Gbps?
Yes
Don't compress (identity) 1.0x | TA=1.00
No
Need high quality? (token agree > 0.95)
Yes
uniform_int8 2.0x | TA=0.97 pplΔ=0.03
No
Moderate quality OK? (token agree > 0.90)
Yes
cachegen 3.5x | TA=0.93 pplΔ=0.81
No
Bandwidth < 10 Gbps? (need max compression)
Yes
kivi_2bit 6.5x | TA=0.74 pplΔ=7.5
No
fp8_e4m3 2.0x | HW-accelerated Best if GPU supports FP8
Avoid for generation tasks:
uniform_int4 TA=0.59 | pplΔ=13k
cascade TA=0.33 | pplΔ=18k
Legend
TA = Token Agreement (greedy decode match rate)
pplΔ = Perplexity delta vs uncompressed
N.Nx = Compression ratio
Break-even: GPU pipelined, Tesla T4
Quality: FP16, 5 models, 50 WikiText prompts

Figure 1: Decision flowchart for KV cache compression in disaggregated serving. Token agreement (TA) and perplexity delta (Δppl) are means over 5 models and 50 WikiText prompts.

2. **Quality-critical applications (TA$>$0.95):** Use uniform INT8. It provides 2$\times$ compression with 97.2% token agreement and negligible perplexity impact (Δppl=0.03).
3. **Moderate quality tolerance (TA$>$0.90):** Use CacheGen for 3.5$\times$ compression with 92.5% token agreement.
4. **Low bandwidth ($<$10 Gbps):** Consider KIVI 2-bit for 6.6$\times$ compression, accepting 73.7% token agreement.
5. **Avoid INT4 and cascade for generation tasks** unless the target model has been validated. Qwen models fail catastrophically under these methods.

6 Limitations

Simulated transfer. Compression and decompression timing is measured on real hardware, but network transfer uses an analytical model ($T = \text{size}/\text{bandwidth}$) validated against TCP localhost measurements. Real networks exhibit protocol overhead, congestion, and variable latency not captured here.

Reimplemented compressors. Our implementations follow published algorithms in NumPy/PyTorch but may differ from original CUDA kernels in absolute speed. Compression ratios and quality metrics, which are our primary focus, are implementation-independent.

Models up to 8B. We evaluate models from 3B to 8B parameters. While KV cache structure scales predictably, disaggregated serving typically targets larger models (70B+) where our absolute timing numbers would differ, though relative compressor rankings should hold.

No batching. We evaluate single-request compression. Batched KV caches could amortize overhead differently, potentially shifting break-even points.

7 Related Work

KV cache compression. KIVI [3] proposes asymmetric 2-bit quantization with per-channel keys and per-token values. KVQuant [5] uses non-uniform quantization with outlier handling. CacheGen [4] applies delta encoding for streaming KV transfer. HACK [6] performs homomorphic computation on quantized caches to avoid decompression. Each paper evaluates against 2–3 baselines under different conditions, making cross-method comparison impossible.

Compression benchmarks. "What Must We Give in Return?" [9] benchmarks 10+ methods but measures only quality, not transfer cost. KVPress [8] evaluates 25+ token eviction methods for memory reduction—no transfer evaluation. "Rethinking KV Cache Compression" [10] critiques existing benchmarks for ignoring throughput under FlashAttention, but focuses on single-GPU serving, not disaggregated transfer.

Disaggregated serving. DistServe [1] and Splitwise [2] establish the prefill/decode disaggregation paradigm but handle KV transfer via raw RDMA without compression. NVIDIA Dynamo integrates KV routing into production serving. None systematically evaluate compression for the transfer bottleneck.

8 Conclusion

KVShuttle provides the first transfer-oriented benchmark of KV cache compression for disaggregated LLM serving. Our evaluation of 14 strategies across 6 models reveals that: the best compressor depends on bandwidth and model architecture; GPU acceleration is essential for compression to be beneficial at datacenter speeds; and cosine similarity is an unreliable quality proxy. We release our toolkit at `https://github.com/awneesht/KVShuttle` to enable reproducible evaluation of future compression methods.

References

[1] Yinmin Zhong, Shengyu Liu, Junda Chen, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *OSDI*, 2024.

[2] Pratyush Patel, Esha Choukse, Chaojie Zhang, et al. Splitwise: Efficient generative LLM inference using phase splitting. In *ISCA*, 2024.

[3] Zirui Liu, Jiayi Yuan, Hongye Jin, et al. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *arXiv preprint arXiv:2402.02750*, 2024.

[4] Yuhan Liu, Hanchen Li, Yihua Cheng, et al. CacheGen: KV cache compression and streaming for fast large language model serving. In *SIGCOMM*, 2024.

[5] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, et al. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *NeurIPS*, 2024.

[6] Jiayi Yao, Hanchen Li, Yuhan Liu, et al. HACK: Homomorphic acceleration of KV cache transfer for disaggregated LLM inference. In *SIGCOMM*, 2025.

[7] Chi-Chih Chang, Wei-Cheng Lin, et al. Palu: Compressing KV-cache with low-rank projection. In *ICLR*, 2025.

[8] Simon Jegou et al. KVPress: Efficient KV cache compression library. NVIDIA, 2025. `https://github.com/NVIDIA/kvpress`.

[9] Jiayi Yuan et al. KV cache compression, but what must we give in return? A comprehensive benchmark of long context capable approaches. *arXiv preprint arXiv:2407.01527*, 2024.

[10] Various authors. Rethinking KV cache compression for large language models. In *MLSys*, 2025.

[11] Tim Dettmers, Mike Lewis, Younes Belkada, Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *NeurIPS*, 2022.

[12] Zhenyu Zhang, Ying Sheng, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, 2024.