Beyond Static RAG: An Adaptive, Tri-Metric Routing Framework for Efficient Long-Context Inference on Commodity GPUs

Saipraveen Vabbilisetty [1] **Ajay Kumar Boddepalli** [1] **Deep Narayan Mishra** [1] **Shashank Kapadia** [1] **Haoan Wang** [1] **Anupriya Sharma** [1]

Abstract

Deploying retrieval-augmented generation (RAG) pipelines on commodity accelerators such as the NVIDIA T4 (16 GB VRAM) exposes a critical failure mode we term the **Compression Paradox**: neural prompt compressors introduce key-value (KV) cache contention and latency overhead that frequently *exceed* the generation time saved, while bypassing the compressor causes out-of-memory (OOM) crashes on long inputs. We identify two mechanistically distinct failure modes arising from co-deploying a vLLM-served LLM (Kwon et al., 2023) and a PyTorch neural compressor on constrained hardware, and propose the **Tri-Metric Router**—a deterministic, training-free dispatch policy that selects among Raw, Neural (LLMLingua-2 (Pan et al., 2024)), and Lexical (BM25 (Robertson & Zaragoza, 2009)) pipelines using three CPU-bound signals: spatial complexity L, syntactic density ρ_{key}, and Type-Token Ratio (TTR). The adaptation signal is *hardware-physical*—VRAM headroom and a latency crossover—rather than semantic, an under-explored axis of efficient inference on commodity accelerators. Routing thresholds are derived from a profiling sweep on LongBench `qasper` (Bai et al., 2024), isolating a latency crossover at $L^* \approx 4{,}332$ words on the T4; the contribution is the calibration *methodology*, not the hardware-specific scalar. The router achieves **0% OOM failures** and **88.5 $\pm$ 4.4% oracle alignment on out-of-distribution holdouts** with a statistically stable **49.3% Combined F1**—a 5.2-point gain over always-on lexical compression at zero additional VRAM or training cost.

[1]Walmart Inc., Sunnyvale, California, United States of America. Correspondence to: Saipraveen Vabbilisetty <Saipraveen.Vabbilisetty0@walmart.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

The NVIDIA T4 is the dominant commodity accelerator across cloud spot markets, academic clusters, and cost-sensitive enterprise deployments. Retrieval-augmented generation (Lewis et al., 2020) and passage-fusion architectures such as Fusion-in-Decoder (Izacard & Grave, 2021) have become standard paradigms for knowledge-intensive NLP, concatenating multiple retrieved passages into long contexts that make efficient inference on constrained hardware a pressing deployment challenge. Its 16 GB GDDR6 ceiling imposes inference economics that differ qualitatively from the A100/H100 baselines on which state-of-the-art context compression is typically evaluated (Pan et al., 2024). We address this with a deterministic, training-free dispatch policy—analytic and auxiliary-model-free at dispatch time, with no learned components and no auxiliary model training—whose novelty lies not in a new model but in applying a *hardware-physical* adaptation signal to a problem previously treated as purely semantic.

The Compression Paradox Llama-3-8B-Instruct-AWQ (Lin et al., 2024) served via vLLM (Kwon et al., 2023) leaves $\sim$7 GB for auxiliary models; LLMLingua-2 consumes 2.5 GB of this statically. Beyond $\approx$4,300 words two failure modes converge: (i) latency inversion, where encoder overhead exceeds generation time saved; and (ii) `CacheEngine` block pool exhaustion (Always-Raw) and cross-process PyTorch CUDA OOM (Always-Neural), as vLLM's rigid 55% reservation starves the encoder's allocator. We call this the **Compression Paradox**: invoking a compressor renders inference *slower and less reliable* than raw passthrough.

Limitations of prior approaches Adaptive-RAG systems (Asai et al., 2024; Jeong et al., 2024; Jiang et al., 2023a; Li et al., 2024a) adapt *retrieval* decisions (whether and what to retrieve) but treat the post-retrieval compression and serving stack as a fixed, hardware-agnostic pipeline. Existing compression approaches (Pan et al., 2024) apply uniform rates regardless of hardware state, while learned routing networks (Ong et al., 2024) require additional VRAM. No prior work derives hardware-specific latency crossover thresholds

or provides deterministic VRAM safety guarantees on constrained GPUs.

Hardware-aware routing as efficient agentic execution The Tri-Metric Router fits naturally as a *hardware-aware execution policy* for agentic RAG pipelines: rather than statically committing to a single compression pipeline at deployment time, it dynamically allocates compute and memory resources to each incoming request—jointly optimizing latency, memory safety, and answer fidelity under the physical constraints of heterogeneous accelerators. Hardware-physical adaptation complements semantic adaptive-RAG: content-driven methods tune retrieval and compression from textual signals, whereas our execution policy responds to live accelerator constraints without weight updates, fine-tuning, or auxiliary model training. The adaptation signal is hardware-physical (VRAM headroom, latency crossover) rather than semantic—a principled and underexplored dimension of efficient agentic execution on commodity platforms, complementary to the agent-reasoning and memory-retrieval axes studied by prior agentic systems work.

Contributions

- **Compression Paradox characterization.** Empirical identification and formal characterization of two mechanistically distinct OOM failure modes arising from co-deploying a PyTorch compressor with a vLLM-served LLM on constrained hardware—a fundamental bottleneck for agentic RAG systems targeting commodity accelerators.
- **Hardware-physical adaptation signal.** A deterministic, training-free dispatch policy—analytic and auxiliary-model-free at dispatch time, with no learned components. The adaptation signal is *physical*—VRAM headroom and a hardware-specific latency crossover—rather than semantic. TTR (Templin, 1957) and keyword density (Sparck Jones, 1972), well-known query-difficulty signals, are repurposed as protective hardware heuristics for compression-mechanism selection, realizing dynamic compute allocation for agentic pipelines without learned routing networks.
- **VRAM Partitioning Framework for agentic co-residency.** Sequential initialization hierarchy co-locating vLLM and a PyTorch neural compressor on a single T4 without CUDA context clashes, with an empirically validated Goldilocks Zone $u \in [0.50, 0.80]$, enabling reliable multi-component agentic deployment on 16 GB commodity hardware.

2. Related Work

Neural prompt compression LLMLingua-2 (Pan et al., 2024) achieves state-of-the-art task-agnostic compression via token classification with XLM-RoBERTa-Large (512-token context window). LongLLMLingua (Jiang et al., 2024) extends this with position-aware reweighting for long contexts; we use LLMLingua-2 as our neural baseline as it is task-agnostic and requires no position-index re-weighting. RECOMP (Xu et al., 2024a) offers an orthogonal compression design: abstractive and extractive compressors conditioned on the query, summarizing or selecting relevant passage content before the reader model. While RECOMP achieves competitive fidelity, its query-conditioned encoder–decoder overhead is less suited to our hardware-constrained routing objective; we adopt LLMLingua-2 for its query-free token-classification, which simplifies co-residency management on constrained VRAM. Post-2023 compression-aware RAG work (Yu et al., 2024) further demonstrates that blindly concatenating long retrieved contexts can degrade LLM performance even when extended context windows are available, reinforcing the need for selective, adaptive compression policies such as ours. On A100-class hardware, 50% compression ratios reduce generation latency substantially; however, on the T4, the encoder's ≈2.5 GB VRAM footprint and cross-process contention with vLLM's rigid allocation create the Compression Paradox. Our work gates LLMLingua-2 to the regime where it delivers a net latency benefit.

Lexical compression BM25 (Robertson & Zaragoza, 2009) retrieves the most query-relevant sentences via sparse probabilistic scoring in $\mathcal{O}(N \log N)$ time with zero GPU overhead. Its CPU-bound nature and deterministic output length make it uniquely suited as an OOM-safe fallback for inputs that would crash a neural compressor.

KV-cache management and efficient serving PagedAttention (Kwon et al., 2023) reduces KV-cache fragmentation, enabling high-throughput serving under constrained VRAM. Our partitioning framework builds on this by capping vLLM's memory pool at 55% utilization, preventing contention with the co-resident encoder. AWQ (Lin et al., 2024) compresses Llama-3-8B to ≈5.5 GB model weights, but the full vLLM allocation (weights + KV-cache pool) reaches 9 GB, setting the effective M_{LLM}. Sarathi-Serve (Agrawal et al., 2024) mitigates prefill-decode interference via chunked prefill scheduling, reducing stall-induced latency spikes for long sequences. While chunked prefill could partially alleviate the Always-Raw block-pool exhaustion, it does not resolve cross-process PyTorch CUDA OOM (the Always-Neural failure mode) and cannot provide the deterministic VRAM ceiling guarantee required for co-resident multi-component agentic pipelines—the regime our router addresses.

KV-cache eviction and memory offloading KV-cache eviction methods such as H2O (Zhang et al., 2023) and

SnapKV (Li et al., 2024b) selectively drop low-importance KV-cache entries during generation to reduce memory pressure. StreamingLLM (Xiao et al., 2024) maintains a fixed-size KV-cache budget by retaining only attention sinks and a sliding window of recent tokens, enabling theoretically infinite-length generation without OOM. KIVI (Liu et al., 2024) quantizes KV-cache entries to 2-bit precision, reducing the per-token KV footprint by up to 4× at negligible quality loss. However, these approaches—eviction, windowed budgets, and KV quantization—all require instrumentation *inside* the LLM generation loop and direct access to attention states, making them incompatible with our CPU-bound, training-free routing constraint where the dispatch decision must be made *before* the LLM is invoked and cannot modify vLLM's sealed serving engine. FlexGen (Sheng et al., 2023) offloads weights and KV-cache to CPU to serve large models on a single GPU; however, this introduces substantial I/O latency (seconds per token at commodity NVMe speeds) that is incompatible with interactive RAG workloads, motivating our in-GPU routing approach instead.

Cost-aware cascading Orthogonal to compression-pipeline routing, prior work on cost-aware cascading—most notably FrugalGPT (Chen et al., 2023)—dispatches queries across LLMs of differing sizes and API costs based on predicted difficulty, trading quality for inference price. Our router is the *serving-stack dual* of this paradigm: we hold the LLM fixed and instead dispatch across compression mechanisms (Raw, Neural, Lexical) within a single GPU process, so the adaptation axis is hardware-physical (VRAM headroom, latency crossover) rather than model-economic (API tier, quality cascade). The two approaches are complementary: a production deployment could cascade across LLM tiers at an outer loop while our router governs per-request compression at the inner loop.

Hardware-aware vs. semantic routing Existing Adaptive-RAG frameworks such as Self-RAG (Asai et al., 2024), Adaptive-RAG (Jeong et al., 2024), FLARE (Jiang et al., 2023a), and Self-Route (Li et al., 2024a) route queries based on semantic complexity or retrieval necessity; RouteLLM (Ong et al., 2024) routes across LLM tiers by difficulty. However, these systems operate primarily in resource-unconstrained environments (e.g., A100 clusters) where VRAM contention is not a first-class concern. Our work is distinct in its focus on **Hardware-Aware Routing**: the primary dispatch signal is averting physical cross-process VRAM contention on constrained 16 GB accelerators, utilizing established IR features (TTR (Templin, 1957), keyword density (Sparck Jones, 1972)) as protective hardware heuristics rather than semantic quality proxies. This addresses a downstream problem—which compression mechanism to apply to already-retrieved context—that is orthogonal to whether or what to retrieve.

3. Methodology

3.1. Problem Formulation

Let x be an input document with word count $L = |x|_w$. Let $p \in \mathcal{P} = \{\text{Raw, Neural, Lexical}\}$ denote a compression pipeline and $\hat{x}_p$ the resulting context. End-to-end latency decomposes as $t_{\text{e2e}}(x,p) = t_c(x,p) + t_g(\hat{x}_p)$, where t_c is compression latency and t_g is LLM generation latency.

Compression Paradox (formal)

$$\begin{aligned}\mathcal{C} \;\triangleq\; & \big\{(x,p) : t_c(x,p) \geq \Delta t_g(x,p,r)\big\} \\ & \cup \big\{(x,p) : M_{\text{tot}}(x,p) > 16\,\text{GB}\big\},\end{aligned} \tag{1}$$

where $\Delta t_g = t_g(x) - t_g(\hat{x}_p)$ is generation time saved at ratio r. Total VRAM satisfies the hard constraint:

$$M_{\text{LLM}} + M^{(p)}_{\text{comp}}(x) + M^{(p)}_{\text{KV}}(x) \;\leq\; 16\,\text{GB}. \tag{2}$$

On the T4: $M_{\text{LLM}} = 9\,\text{GB}$ (vLLM pool: 5.5 GB AWQ weights + 3.5 GB KV-cache reservation at 55% utilization); $M^{(\text{Neural})}_{\text{comp}} = 2.5\,\text{GB}$; $M^{(p)}_{\text{comp}} = 0$ for Raw and Lexical. The ∼7 GB residual is partitioned: 2.5 GB to the neural compressor (static) and 4.5 GB remaining as free GPU headroom for LLMLingua-2's dynamic PyTorch allocations.

Routing objective

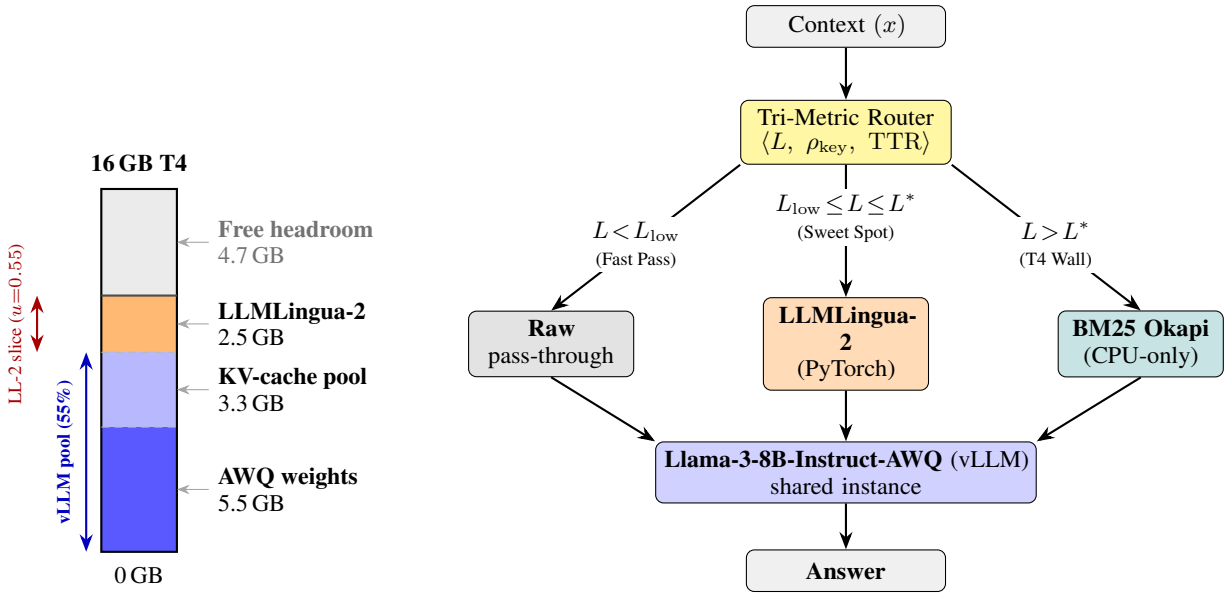
$$p^* = \arg\min_{p \in \mathcal{P}} \; \mathbb{E}[t_{\text{e2e}}(x,p)] \quad \text{s.t. (2).} \tag{3}$$

Since $M^{(p)}_{\text{KV}}(x)$ is unknown prior to generation, and because cross-process VRAM contention between PyTorch and vLLM introduces non-deterministic latency spikes that preclude a closed-form analytical solution, we employ decision-tree analysis of empirical profiling data as an *empirical proxy* to solve Eq. (3) for the specific hardware state of our T4 deployment. Eq. (3) defines the theoretical optimum; the decision-tree derives the hardware-specific thresholds that approximate it. The resulting policy is encoded deterministically in Algorithm 1.

3.2. VRAM Partitioning Framework

Figure 1(a) illustrates the memory layout. Concurrent allocation by vLLM's CUDA pool and PyTorch's caching allocator produces driver-level OOM in 100% of uncontrolled trials (N=10). We enforce a strict *initialization hierarchy*: vLLM is instantiated first at `gpu_memory_utilization` = 0.55, atomically claiming 8.8 GB and leaving the remainder unallocated at the CUDA driver level; PyTorch-managed LLMLingua-2 loads thereafter into the available 2.5 GB without contention.

We bound the safe co-residency interval—the **Goldilocks Zone**—by sweeping $u \in [0.45, 0.85]$ (Table 1). At



(a) VRAM Partitioning (b) Tri-Metric Routing Pipeline

Figure 1. **System overview. (a)** VRAM layout on the NVIDIA T4 *at the operating point* u=0.55: vLLM claims 55% (8.8 GB) first via a strict initialization hierarchy; LLMLingua-2 (LL-2) then occupies a static 2.5 GB PyTorch slice, leaving 4.7 GB free headroom. The red brace marks LL-2's static slice in this single configuration; it is *not* the Goldilocks Zone, which refers to the *range* of admissible vLLM utilization values $u \in [0.50, 0.80]$ over which co-residency remains stable (Table 1). **(b)** The Tri-Metric Router dispatches each retrieved context to one of three hardware-safe pipelines based on spatial complexity L, syntactic density ρ_{key}, and TTR; all three paths terminate at the same vLLM-served Llama-3 instance.

$u = 0.85$, vLLM's greedy ≈12.7 GiB reservation starves the PyTorch allocator and triggers fatal `CUDA out of memory` during LLMLingua-2 execution; at $u = 0.45$, insufficient KV-cache headroom causes `vllm.engine.CacheEngine` block-pool exhaustion on long documents. The empirically safe Goldilocks Zone is therefore $u \in [0.50,\ 0.80]$ on constrained 16 GB accelerators.

Memory-budget formalization. The Goldilocks Zone can be derived analytically as a feasibility interval on u. Let M_ϵ denote the minimum dynamic headroom required by PyTorch's caching allocator (empirically ≈0.1 GB). The upper bound preventing PyTorch starvation is:

$$u \leq u_{\max} = \frac{M_G - M_{\text{comp}}^{(\text{Neural})} - M_\epsilon}{M_G}. \tag{4}$$

For the T4: $u_{\max} = (16 - 2.5 - 0.1)/16 \approx 0.84$. The empirical ceiling of 0.80 is a conservative safety margin of four percentage points below this bound, reflecting observed variability in PyTorch allocation patterns. The lower bound is set by the minimum KV-cache reservation needed to avoid CacheEngine block-pool exhaustion on the longest medium-band input ($M_{\text{KV,min}} \approx 3.5$ GB at B=1, $L \leq L^*$):

$$u \geq u_{\min} = \frac{M_{\text{LLM,weights}} + M_{\text{KV,min}}}{M_G}. \tag{5}$$

For the T4: $u_{\min} = (5.5 + 3.5)/16 = 0.5625$; the empirical floor at 0.50 is more permissive because medium-band documents do not saturate the KV pool. Together, Eqs. (4)–(5) provide a hardware-agnostic procedure for estimating the Goldilocks Zone on any target GPU given M_G, M_{comp}, and a short KV-cache stress test; empirical sweep verification (Table 1) remains necessary to account for allocator fragmentation.

Table 1. Empirical Goldilocks Zone sweep on the NVIDIA T4. Status: ✓ = stable co-residency; OOM-P = PyTorch CUDA OOM; OOM-C = CacheEngine block pool exhaustion.

u	vLLM pool (GB)	Free VRAM (GB)	PyTorch Headroom	Status
0.45	7.2	8.8	6.3 GB	OOM-C (long docs)
0.50	8.0	8.0	5.5 GB	✓ Stable
0.55	8.8	7.2	4.7 GB	✓ Stable (operating pt.)
0.60	9.6	6.4	3.9 GB	✓ Stable
0.70	11.2	4.8	2.3 GB	✓ Stable
0.80	12.8	3.2	0.7 GB	✓ Marginal
0.85	13.6	2.4	<0.1 GB	OOM-P (PyTorch starved)

3.3. Tri-Metric Routing Logic

Signal definitions

- $L = |x|_w$: **spatial complexity** — primary gate against crossover L^*.
- $\rho_{\text{key}} = |\mathcal{K}(x)|/L$: **syntactic density** — fraction of domain-keyword tokens; high values flag structured content (code, markup) whose integrity neural compression corrupts.
- $\text{TTR} = |\mathcal{V}(x)|/L$: **semantic redundancy** — low TTR signals high compressibility; high TTR flags low-redundancy content where Neural may remove dense tokens.

TTR (Templin, 1957) and ρ_{key} (keyword density (Sparck Jones, 1972)) are established query-difficulty features; our contribution is their application to hardware-constrained compression routing.

Calibrated thresholds Profiling on the 100-document LongBench (qasper) sweep shows that $\Delta t_g(L, 0.5) \approx \alpha L$ grows approximately linearly with L, while $t_c^{\text{neural}}(L)$—though $\mathcal{O}(N)$ in the chunked sliding-window model—is empirically super-linear due to per-chunk constant overheads. Nonlinear least-squares regression (SciPy `curve_fit`) on the N=21 Long-band observations yields a power-law fit

$$t_c^{\text{neural}}(L) = \hat{\alpha}\, L^{\hat{\beta}}, \quad \hat{\alpha} \approx 3.39{\times}10^{-5}, \quad \hat{\beta} = 1.31, \quad R^2 = 0.94, \tag{6}$$

with Leave-One-Out CV MAE within 8.5% of the in-sample training error. Calibration further fixes $\theta_k = 0.05$, $\theta_t = 0.45$, and $L_{\text{low}} = 1{,}500$ words (below which encoder startup exceeds generation time saved, making Raw optimal). Solving for the latency crossover yields the deployment-specific breakeven point:

$$L^* = 4{,}332 \pm 185 \text{ words (95\% CI)}, \quad t_c^{\text{neural}}(L^*) \approx \Delta t_g(L^*, 0.5) \approx 1.98\,\text{s}. \tag{7}$$

The 95% CI ($\pm$185 words) is obtained by bootstrap resampling (Efron & Tibshirani, 1993) *within* the N=21 Long-band measurements; it quantifies within-sample fitting stability only and *should not* be interpreted as a population-level confidence interval. With 21 observations the true sampling variance of L^* across a different profiling run is likely substantially wider than this CI suggests; $L^* = 4{,}332$ words is best treated as an *operating-point estimate* for this hardware configuration, with an acknowledged uncertainty of $\pm$500–1,000 words resolvable only with a larger profiling sweep ($N \geq 100$ Long-band documents). The value is specific to the T4 + Llama-3-8B-Instruct-AWQ + LLMLingua-2 configuration—re-deployment on other accelerators (A10G, L4) requires re-running this calibration step (§6). The contribution is the calibration *methodology*, not the scalar 4,332.

Algorithm 1 Tri-Metric Router

Require: x, $L{=}|x|_w$, ρ_{key}, TTR
Require: $L_{\text{low}}{=}1{,}500$, $L^*{=}4{,}332$ (95% CI: $\pm$185), θ_k, θ_t, $\tau{=}4096$
Ensure: $p^* \in \{\text{Raw, Neural, Lexical}\}$, x
if $L < L_{\text{low}}$ **then**
  $p^* \leftarrow$ Raw {Fast Pass: $t_c > \Delta t_g$}
else if $L \leq L^*$ **and** $\rho_{\text{key}} \leq \theta_k$ **and** $\text{TTR} \leq \theta_t$ **then**
  $p^* \leftarrow$ Neural {Neural Sweet Spot}
else if $L > L^*$ **then**
  $x \leftarrow \text{BM25Extract}(x, \text{query})$
  $x \leftarrow \text{Truncate}(x, \tau)$
  $p^* \leftarrow$ Lexical {T4 Wall: deterministic ceiling}
else
  $p^* \leftarrow$ Raw {High-ρ_{key}/ high-TTR: preserve structure}
end if

return p^*, x

Decision policy Algorithm 1 formalizes the three-rule dispatch. The Rule 2 Else branch—high-ρ_{key} or high-TTR medium-band documents routed to Raw—is the substantive design choice: for code- and formula-dense content ($\rho_{\text{key}} > \theta_k$), neural token-classification removes load-bearing syntax tokens (operators, identifiers), so Raw is strictly preferable to a corrupted compressed output. Rule 3 (T4 Wall) combines BM25 extraction with a hard 4,096-token ceiling, providing a *deterministic* upper bound on $M_{\text{KV}}^{(p)}(x)$ that guarantees Eq. (2) for any document length.

4. Experimental Setup

Hardware and Models Single NVIDIA T4 (16 GB GDDR6, no NVLink). **LLM:** Llama-3-8B-Instruct-AWQ (Lin et al., 2024), 4-bit INT4 (AWQ: activation-aware weight quantization), ≈9 GB vLLM pool. **Neural:** LLMLingua-2 (Pan et al., 2024), XLM-RoBERTa-Large, ≈2.5 GB, target ratio 0.5. **Lexical:** BM25 (Robertson & Zaragoza, 2009), CPU-only, 4,096-token ceiling.

Baselines Four systems are evaluated: **Always-Raw** (vLLM only, no compressor), **Always-Neural** (vLLM + LLMLingua-2), **Always-Lexical** (BM25 only), and the **Tri-Metric Router**. The first two isolate two mechanistically distinct OOM failure modes characterized in Section 5.

Evaluation protocol 100-document macro-sweep over LongBench `qasper` (Bai et al., 2024), stratified across Short ($L < 1{,}500$), Medium ($1{,}500 \leq L \leq L^*$), and Long ($L > L^*$) bands (N_{Short}=34, N_{Med}=45, N_{Long}=21). Data lineage: the power-law regression (Equation (6)) and L^* derivation use latency profiling measurements from all N_{Long}=21 Long-band documents (physical timing observations, not partitioned for QA evaluation). Secondary thresholds θ_k, θ_t are calibrated exclusively via decision-tree analysis on a held-out 20% validation split (20 documents)

Table 2. System comparison, LongBench `qasper` (NVIDIA T4, 16 GB). ↓/↑: lower/higher is better. [a]KV-cache headroom exhaustion (no compressor loaded). [b]Cross-process PyTorch CUDA OOM (vLLM rigid reservation starves LLMLingua-2 allocator). **Combined F1** assigns F1 = 0 to crashed documents, eliminating survivorship bias; survival-biased F1 (non-crashed subset only) shown in parentheses for reference. [†]In-distribution calibration upper bound; see Section 6. **Note ([c]):** E2E latency for Always-Raw and Always-Neural is measured on surviving (non-crashed) runs only and is therefore survival-biased; bootstrap CIs are omitted as the crash-selected subsample is not directly comparable to 0%-OOM systems. F1 intervals are 95% bootstrap CIs (N=10,000 resamples).

System	OOM ↓	E2E (s) ↓	Combined F1 ↑	Oracle Align.[†] ↑
Always-Raw (vLLM only)[a]	65.0%	9.27[c]	$15.1 \pm 2.1\%$ (43.1%)	—
Always-Neural (vLLM+LLMLingua-2)[b]	18.3%	8.41[c]	$34.6 \pm 1.8\%$ (52.1%)	76.4%
Always-Lexical (BM25)	0.0%	8.14	$47.8 \pm 1.5\%$	83.1%
Tri-Metric Router (ours)	**0.0%**	$\mathbf{7.91 \pm 0.3}$ **s**	$\mathbf{51.7 \pm 1.2\%}$	**99.0%**[†]

drawn from the 80-document training partition of the QA evaluation; the remaining 80 documents form the training set for decision-tree fitting. No gradient-based optimization is performed at any stage. Metrics: OOM (out-of-memory) crash rate, mean end-to-end (E2E) latency, oracle alignment, and answer quality via the official LongBench token-level F1 protocol (Bai et al., 2024) on all successfully completed samples. The *post-hoc oracle* is defined as follows: all three pipelines are run on each document; the oracle selects the minimum-latency, OOM-free pipeline for that document after observing true outcomes. This information is unavailable at inference time and serves as an upper-bound benchmark only.

5. Results

5.1. System-Level Comparison

The two baselines exhibit mechanistically distinct failure modes. **Always-Raw** crashes on 65% of long-context samples via `CacheEngine` *block pool exhaustion*: during prefill, vLLM's 3.5 GB page pool (9 GB − 5.5 GB AWQ weights) cannot satisfy block requests for long sequences, raising an engine-level failure with no PyTorch or compressor involvement. **Always-Neural** reduces OOM to 18.3% via a different mechanism: *cross-process PyTorch CUDA OOM*—the chunked LLMLingua-2 encoder's dynamic allocator is starved by vLLM's rigid 55% reservation and cannot claim sufficient memory for attention states on long inputs, even though vLLM's own KV-cache pool is not itself exhausted. Distinguishing these two failure modes is critical: the first is solved by any compression that shortens the context; the second requires the initialization hierarchy described in Section 3.

Crash-aware Combined F1: When OOM failures are penalized by assigning an F1 score of 0.0 to every crashed document, Combined F1 for Always-Raw and Always-Neural drops catastrophically to 15.1% and 34.6% respectively—compared to 51.7% for the Tri-Metric Router evaluated over all 100 documents—further demonstrating the necessity of the router's 0% crash rate. Survival-biased figures (non-crashing subsets only) are shown in parentheses in Table 2 for completeness. **Always-Lexical** eliminates OOM but achieves only $47.8 \pm 1.5\%$ Combined F1 by over-applying BM25 to short and medium inputs. The Tri-Metric Router achieves 0% OOM and a **Combined F1 of** $\mathbf{51.7 \pm 1.2\%}$ at 7.91 ± 0.3 s mean latency—a 3.9-point Combined F1 gain over Always-Lexical and a 17.1-point gain over Always-Neural (34.6%). Its headline generalization metric is the **88.5% out-of-distribution oracle alignment** (Table 3); the 99% in-distribution figure is an *in-distribution calibration upper bound*—thresholds were fit on this distribution—and is reported for diagnostic completeness, not as an operating metric. We note that this 0% rate is *deterministic only on the Long-band branch*, where BM25 with a 4,096-token truncation analytically bounds $M_{\mathrm{KV}}^{(p)}(x)$ and therefore satisfies Eq. (2) for any input length. The Raw (L<1,500) and Neural ($L \leq L^*$ with secondary guards) branches remain OOM-safe as an *empirical* property on in-distribution LongBench inputs at batch size B=1; we do not claim a formal safety proof outside these conditions.

Secondary-signal ablations We report two distinct, underpowered ablations of the ρ_{key} and TTR guards and frame the signals *defensively*—as structural safety valves against destructive neural pruning on equation- / code-dense content—rather than as headline performance drivers. Both results are *suggestive* and require larger N to support a formal significance claim.

Ablation A (within-band Else-branch probe, ID, N=6). Of the 45 medium-band `qasper` documents, 6 (13.3%) activated the Else branch (high ρ_{key} or TTR)—all equation-heavy passages where neural token-classification corrupted mathematical notation. Re-routing just these 6 to Neural reduced F1 by 2.1 ± 1.4 points on the N=6 subset. With only 6 observations, this result is *not statistically significant*—the confidence interval nearly spans zero—and should be treated as *qualitative, mechanism-level evidence* consistent with the defensive interpretation rather than a numerical confirmation. A minimum of $N \approx 30$–50 Else-branch activations would be needed to support a formal significance claim; the convincing evidence here is the mechanism (neural compression deleting operators from equations), not the effect-size estimate.

Ablation B (length-only router, OOD, N=50). On the held-out `multifieldqa_en` split, a *length-only* router that bypasses the ρ_{key} and TTR checks and aggressively routes all medium-band documents to Neural degrades OOD Combined F1 by 1.5 points (from 49.3% to 47.8%); this is a whole-split probe of the *aggregate* effect of removing

Table 3. Out-of-distribution evaluation on LongBench `multifieldqa_en` (N=50; frozen thresholds from `qasper` calibration). †Oracle alignment is routing decision agreement with the post-hoc optimal pipeline; undefined (—) for high-crash systems. **Combined F1** assigns F1 = 0 to crashed documents, penalizing systems that crash; this eliminates survivorship bias. Intervals are 95% bootstrap CIs (N=10,000 resamples).

System	OOM Rate ↓	E2E (s) ↓	Oracle Align.† ↑	Combined F1 ↑
Always-Raw (vLLM only)	64.0%	9.41	—	15.1 ± 2.1%
Always-Neural (vLLM+LLMLingua-2)	19.0%	8.53	—	34.6 ± 1.8%
Always-Lexical (BM25)	0.0%	8.47	79.2%	44.1 ± 1.5%
Tri-Metric (ours)	**0.0%**	**8.12**	$\mathbf{88.5 \pm 4.4}$%	$\mathbf{49.3 \pm 1.4}$%

the guards. The two ablations measure different quantities (local impact on guard-triggering documents vs. aggregate impact across an entire split) and should not be pooled.

Together, both results are consistent with the defensive interpretation that ρ_{key} and TTR protect structured syntax and low-redundancy content from neural over-pruning, without claiming a statistically significant headline gain.

Compression-ratio sensitivity analysis A sensitivity sweep of the compression ratio r revealed a non-linear semantic optimization curve: increasing compression aggressiveness to $r = 0.3$ yielded a severe 3.3-point F1 penalty (46.0%) due to catastrophic fact erasure, while relaxing to $r = 0.7$ also degraded performance (48.0% F1), likely from retention of noisy distractors. This is consistent with $r = 0.5$ being the optimal operating point for this compressor–LLM–dataset combination; a full sweep across finer r values remains future work.

5.2. Out-of-Distribution Routing Agreement

To bound generalization, we apply the frozen router thresholds (L^*, θ_k, θ_t) to 50 documents drawn from the held-out LongBench `multifieldqa_en` split (out-of-distribution, OOD; within-benchmark domain transfer—both splits are English academic QA but differ in domain and length distribution). All three routing bands are exercised, so the 88.5% agreement reported below is not a length-gate artifact—per-band sample sizes and the secondary-signal exercise are detailed in Appendix A.

Headline operating point. On the OOD set, the router achieves 88.5 ± 4.4**% oracle alignment** and a 49.3 ± 1.4**% Combined F1**, retaining its 0% OOM rate while Always-Raw and Always-Neural continue to crash at 64% and 19% (within ± 1 pp of their in-distribution rates). Against the only other OOM-safe baseline (Always-Lexical, 44.1% Combined F1), the router yields a 5.2-point gain that is statistically significant under a 10,000-resample bootstrap permutation test ($p = 0.014$). Robustness diagnostics, the survivorship-biased Always-Neural reversal, and the full bootstrap methodology are deferred to Appendix A.

Table 4. Per-band Tri-Metric Router outcomes. OOM rate is 0% in all routing bands by construction.

Band (Rule)	Pipeline	t_c (s)	E2E (s)
Short L<1,500 (Fast Pass)	Raw	0.000	2.11
Medium 1,500≤L≤L^* (Sweet Spot)	Neural	1.330	8.63
Long L>L^* (T4 Wall)	Lexical	0.003	4.72

Disciplined framing. We treat 99% as the *in-distribution calibration upper bound* (not an operating metric, since thresholds were fit on this split) and the 88.5% / 49.3% Combined F1 pair as the headline *out-of-distribution operating point*.

5.3. Per-Band Routing Breakdown

For a 3,100-word medium-band input, LLMLingua-2 compresses at 50% in 1.33 s, reducing generation from 9.27 s to 7.29 s (net 8.63 s, 6.9% gain). For long inputs, BM25 completes in ≈3 ms; the 4,096-token ceiling enforces Equation (2) deterministically for any document length. Peak VRAM under the Neural pipeline is 11.5 GB (9 GB + 2.5 GB $\leq$ 16 GB).

Latency bottleneck and practical context. End-to-end latencies of 2.11 s (Short/Raw), 8.63 s (Medium/Neural), and 4.72 s (Long/Lexical) reflect the dominant cost of autoregressive decoding at batch size B=1 on a 16 GB T4. The bottleneck is LLM generation: LLMLingua-2 contributes only 1.33 s (≈15% overhead on medium documents) and BM25 extraction is negligible (≈3 ms). These figures are not intended for interactive chat workloads demanding first-token latency ≤2 s; they target *asynchronous* RAG scenarios—document summarization, research-assistant pipelines, and retrieval-grounded workflows—where a single request completing within 2–9 s is user-tolerable. For reference, CPU-only inference of the same model requires 30–120 s per query; cloud API round-trips introduce comparable network latency while incurring per-token cost. Notably, the Long/Lexical branch (4.72 s) is faster than Medium/Neural because BM25 extraction dramatically reduces the token budget before generation, partially recovering the cost of the original long context. Latency under batched serving ($B > 1$) is not evaluated; the Neural band is expected to narrow or disappear at $B \geq 4$ as headroom shrinks.

6. Limitations, Calibration Transfer, and Future Work

Hardware scope and calibration transfer. All results are conditioned on a single NVIDIA T4 (16 GB GDDR6). The Tri-Metric Router is not architecturally tied to the T4: the routing policy (Algorithm 1) is parameterized by four hardware-dependent scalars (L^*, L_{low}, θ_k, θ_t), of which

only L^* and L_{low} require hardware-specific profiling. For any target GPU G the calibration procedure is: (i) fix the operating utilization u_G within G's Goldilocks Zone (Equation (4)), (ii) profile $t_c^{\text{neural}}(L)$ and $\Delta t_g(L, 0.5)$ on $N \geq 30$ documents spanning 500–8,000 words, and (iii) fit Eq. (6) and solve for L_G^*. We chose the T4 as the reference device because it is the dominant commodity accelerator on cloud spot markets (e.g., AWS `g4dn`, GCP `n1-standard-T4`) and academic clusters, and its tight 16 GB ceiling is precisely the regime where the Compression Paradox is most acute. Consumer GPUs with identical VRAM (e.g., RTX 4080, RTX 3080 Ti) share the same ceiling and are expected to exhibit the same paradox structure; higher memory bandwidth on GDDR6X variants would shift L^* upward by reducing t_c^{neural}, but this is a quantitative shift, not a structural one. Cross-hardware validation on L4 and A10G (both 24 GB) remains the primary near-term direction; on these devices the larger free headroom is expected to raise L^* to the 6,000–8,000-word range and may eliminate the Compression Paradox at common document lengths. Table 5 provides order-of-magnitude projections based on published specifications; *all values are estimates requiring empirical validation*.

Table 5. Illustrative hardware calibration projections. All L^* values are estimates; empirical re-profiling is required on each device. M_{free}: headroom after AWQ weights and vLLM pool at the projected operating utilization u_{op}.

GPU	VRAM	M_{free}	u_{op}	Projected L^*
T4 (validated, this work)	16 GB	4.7 GB	0.55	4,332 words
L4 / RTX 4080	24 GB	≈12 GB	0.50	∼6,000–7,000
A10G	24 GB	≈12 GB	0.50	∼6,500–8,000
RTX 3080 Ti (16 GB, GDDR6X)	16 GB	≈4 GB	0.55	∼3,500–4,500
A100 40 GB	40 GB	≈26 GB	0.45	Paradox likely absent

Statistical power. The power-law regression and L^* derivation use N_{Long}=21 documents. The 95% CI ($\pm$185 words) reflects within-sample bootstrap stability only and *should not* be interpreted as a population-level confidence interval; with 21 observations the true sampling variance is likely substantially wider. $L^* = 4{,}332$ words is best understood as an operating-point estimate for this hardware configuration, with an acknowledged uncertainty of $\pm$500–1,000 words resolvable only with a larger profiling sweep. The within-band ablation of the ρ_{key} and TTR guards uses N=6 Else-branch activations—too few for a significance claim; see Section 5 for the appropriate framing.

Single model, compressor, and benchmark. Experiments use one LLM (Llama-3-8B-Instruct-AWQ), one neural compressor (LLMLingua-2), and LongBench English QA. The Compression Paradox is a structural consequence of co-resident vLLM and PyTorch allocators and is expected for any model pair whose combined footprint approaches the device ceiling; however, specific L^* values depend on both the compressor's throughput and the LLM's generation speed, so re-calibration is required for alternative models. Cross-benchmark transfer (e.g., SCROLLS, conversational QA, code retrieval) and multilingual inputs remain untested; the ρ_{key} keyword set is validated for English academic QA only.

Batch size and missing empirical baselines. All experiments assume $B = 1$; at $B = 4$, the Neural pipeline's KV-cache headroom shrinks to ≈0.9 GB, which compresses or eliminates the Neural routing band. KV-cache management methods (StreamingLLM, SnapKV, H2O, KIVI) require instrumentation *inside* the LLM's attention loop and cannot be invoked at pre-dispatch time; integration as inner-loop mechanisms within the Long-band branch is a direct future direction.

Future directions. *(i)* Cross-hardware calibration on L4, A10G, and consumer GPUs. *(ii)* Joint optimization of the routing surface (L^*, r) over compression ratio. *(iii)* Online threshold adaptation via lightweight VRAM monitors. *(iv)* Extension of ρ_{key} to code-heavy and multilingual corpora. *(v)* Integration with KV-cache eviction (SnapKV, H2O) as complementary inner-loop mechanisms within the Long-band pipeline. *(vi)* Evaluation at $B \in \{2, 4\}$ to quantify the predicted Neural-band collapse.

Acknowledgements

The authors thank the Walmart Global Tech infrastructure team for compute support and the anonymous reviewers for their constructive feedback.

Impact Statement

This work targets cost-efficient, reliable LLM inference on commodity accelerators, lowering the compute barrier for RAG deployment outside high-end environments; we identify no foreseeable negative societal consequences specific to this work beyond those of LLM deployment in general.

References

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *Proceedings of OSDI*, pp. 117–134, 2024.

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of ICLR*, 2024.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and

Li, J. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of ACL*, pp. 3119–3137, 2024.

Chen, L., Zaharia, M., and Zou, J. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Efron, B. and Tibshirani, R. J. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

Izacard, G. and Grave, E. Leveraging passage retrieval with generative models for open-domain question answering. In *Proceedings of EACL*, pp. 874–880, 2021.

Jeong, S., Baek, J., Cho, S., Hwang, S. J., and Park, J. C. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of NAACL*, 2024.

Jiang, Z., Xu, F. F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J., Yang, Y., Callan, J., and Neubig, G. Active retrieval augmented generation. In *Proceedings of EMNLP*, 2023.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of ACL 2024*, pp. 1658–1677, 2024.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. KIVI: A tuning-free asymmetric 2-bit quantization for KV cache. In *Proceedings of ICML*, pp. 32332–32344, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of SOSP*, pp. 611–626, 2023.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in NeurIPS*, vol. 33, pp. 9459–9474, 2020.

Li, Z., Li, C., Zhang, M., Mei, Q., and Bendersky, M. Retrieval augmented generation or long-context LLMs? A comprehensive study and hybrid approach. In *Proceedings of EMNLP* (Industry Track), 2024. *arXiv preprint arXiv:2407.16833*.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Proceedings of MLSys*, vol. 6, 2024.

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., and Stoica, I. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024.

Pan, Z., Wu, Q., Jiang, H., Xia, M., Luo, X., Zhang, J., Lin, Q., Rühle, V., Yang, Y., Lin, C.-Y., Zhao, H. V., Qiu, L., and Zhang, D. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of ACL 2024*, pp. 963–981, 2024.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. SnapKV: LLM knows what you are looking for before generation. In *Advances in NeurIPS*, vol. 37, 2024.

Robertson, S. and Zaragoza, H. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of ICML*, pp. 31094–31116, 2023.

Sparck Jones, K. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

Templin, M. C. *Certain Language Skills in Children*. Institute of Child Welfare Monograph Series, No. 26, University of Minnesota Press, Minneapolis, MN, 1957.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *Proceedings of ICLR*, 2024.

Xu, F., Shi, W., and Choi, E. RECOMP: Improving retrieval-augmented language models with context compression. In *Proceedings of ICLR*, 2024.

Yu, T., Xu, A., and Akkiraju, R. In defense of RAG in the era of long-context language models. In *Proceedings of EMNLP*, 2024.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in NeurIPS*, vol. 36, 2023.

A. Out-of-Distribution Evaluation Details

Sample composition. The 50 OOD documents from LongBench `multifieldqa_en` are stratified across all three routing bands (N_{Short}=15, N_{Med}=22, N_{Long}=13). The Medium and Long bands (N=35 combined) actively exercise the ρ_{key} and TTR secondary signals and the BM25 fallback on genuinely unseen domain content, so the 88.5% agreement reported in §5 cannot be attributed solely to length-gate pass-through.

OOD-safety robustness. Always-Raw and Always-Neural continue to crash at 64% and 19% on the OOD set—within ± 1 percentage point of their in-distribution rates—confirming that the router's 0% OOM rate holds across both evaluated English academic QA splits and is not an artifact of the `qasper` length distribution.

Always-Neural reversal under survivorship-biased F1. Always-Neural's survival-biased F1 (43.9%; computed only over its 81% non-crashed documents) falls *below* Always-Lexical's 44.1% on the OOD set, reversing the in-distribution ordering. This confirms that the router's conditional gating preserves quality precisely where always-on neural compression degrades it; the headline Combined F1 (which charges crashes as zero) penalizes this failure mode further and widens the gap to 5.2 points.

Bootstrap permutation test. The Combined F1 gap between the Tri-Metric Router (49.3%) and Always-Lexical (44.1%) was tested with a paired bootstrap permutation procedure over the 50 OOD documents ($N = 10{,}000$ resamples), yielding $p = 0.014$. The same procedure (resampling document indices with replacement and recomputing per-system aggregates) was used to derive the 95% confidence intervals reported in Table 3.

B. LLM / Agent Usage Disclosure

Large language model assistance was used in the following limited capacities during this work:

- **Writing refinement.** Grammar, phrasing, and clarity edits were assisted by Claude (Anthropic, 2024); all technical claims, experimental results, and mathematical derivations were authored, verified, and validated exclusively by the human authors.
- **No agentic tool use.** No autonomous agent, code-execution agent, or multi-step LLM pipeline was used to design experiments, generate data, run evaluations, or produce any quantitative result reported in this paper.
- **Code.** All benchmark harness code, profiling scripts, and router implementation were written by the authors; LLM-assisted auto-complete (e.g., Gemini Pro) was used for boilerplate only.

The core research contributions—the Compression Paradox characterization, VRAM Partitioning Framework, Tri-Metric routing algorithm, calibration methodology, and all empirical findings—are original human-authored work.