WHEN THE SECOND GPU HELPS: TOPOLOGY-AWARE MODE SELECTION FOR NARROW-INTERCONNECT LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLM serving assumes NVLink-class interconnects, yet most multi-GPU deployments (consumer workstations, PCIe-only cloud VMs) link cards at a few GB/s. On such a *narrow* link, does a second GPU help, and in which mode? *Replica* places a full model copy per card, roughly doubling throughput when the context fits one card; *staged split* partitions the layers, extending reach but moving the per-token attention state (KV cache, growing with context) over the slow link every step. We show that this decision reduces to a **calibrated discrete decision rule** over **two measurable coefficients**—portable after per-engine recalibration: the per-token transport tax τ and the replica's fixed-overhead gap ΔA. Their ratio gives a crossover context $\mathrm{ctx}^* = \Delta A/\tau$; we are explicit about identifiability: the *decision* is robust to $\pm 15\%$ coefficient error (first flip $\approx 31\%$), while the point estimate of ctx^* is the fit's least-identified parameter (LOCO 120–1,281). We measure both at four link settings across three hosts (6.65/13.3/17.6/25.4 GB/s), four engines, two models, up to $k{=}4$ no-NVLink GPUs: the split tax is sub-linear (flat through $k{=}3$), rejecting the shared-root $(k{-}1)^2$ prediction on distinct-bridge hosts, while replica scaling is near-linear ($\times 3.89$). Because τ is *system-controllable*, the coefficients also serve as an interface: a per-token cost reduction (an fp8-KV plugin, quality-failing in the tested configuration) measurably shifts ctx^* rightward; we identify where the crossover breaks (fast links, $\Delta A{<}0$). The abstraction is embodied in an automatic decision layer that gates on memory, calibrates online, and drives llama.cpp within its 32,768-token admission window while reaching 65,536-token contexts on two 12 GB consumer GPUs through its own 4-bit staged path, with no manual quantization or RoPE tuning; the reach is not exclusive—manual vLLM and ExLlamaV3 configurations also admit 65K, and are faster there.

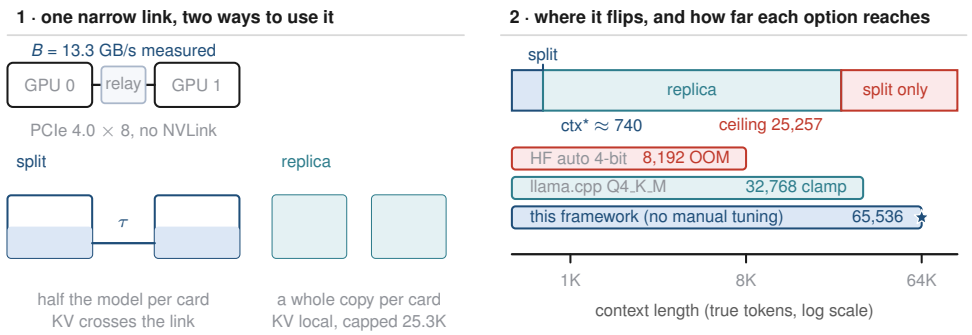


Figure 1: **Two coefficients decide the mode.** A *staged split* partitions the layers and pays a per-token transport tax τ; a *replica* keeps KV local but is capped by the per-card KV ceiling. Measured on the platform, they are $\tau{=}0.37$ ms/tok and $\Delta A{=}284$ ms, putting the crossover at $\mathrm{ctx}^* \approx 740$ tokens (linear law $\Delta A/\tau = 769$). One log context axis below shows the regimes to scale and each option's reach.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Modern LLM inference systems converge on a high-bandwidth assumption—NVLink intra-node, InfiniBand inter-node—that excludes a large deployment population: consumer dual-GPU workstations on motherboard PCIe lanes, cloud VMs without NVLink, USB4/Thunderbolt edge servers, and legacy NVLink-less nodes. In all of these, the bottleneck is scheduling execution, movement, and recovery so the visible cost of narrow interconnects stays below the benefit of additional devices. The question is not "how many devices?" but "given my topology, which mode is optimal?"

This paper asks *when the second accelerator helps, and in which mode?* (Figure 1). We propose a **topology-aware decision layer** for bandwidth-constrained multi-accelerator LLM inference that reads a labeled topology graph (bandwidth, latency, memory) and calibrates three per-mode latency coefficients per engine and session. Its contribution is the pair of measurable coefficients below, τ and ΔA, and their embodiment.

Contributions.

1. A **calibrated discrete mode decision** and a **calibration protocol** for it: per-request single / staged-split / replica by a feasibility gate plus an EWMA-calibrated score, with a noise-band fallback to a measured regime rule; 21/22 grid (23/24 all points), 44/47 off-grid, robust to $\pm 15\%$ coefficient error. The coefficients $(\tau, \Delta A)$ are *calibration coordinates*—measurable, per-engine, explicit about identifiability limits—not universal constants.
2. A **coefficient-as-interface validation**: because τ is system-controllable, a per-token cost reduction (fp8 KV, SGLang) measurably shifts ctx* rightward without any change to the decision layer—the mechanism by which KV-placement/sparse-retrieval systems compose with our layer; the probe is reported with both qualifiers—mechanism (attention reads, not relay bytes) and measured quality cost ($0/48$ vs. $48/48$ bf16).
3. An **automatic decision layer**: it gates memory feasibility, calibrates online, drives llama.cpp inside its 32,768-token admission window, and reaches 65,536-token contexts on two 12 GB consumer GPUs through the framework's own 4-bit staged path, with zero manual quantization or RoPE tuning (Llama-3-8B at 65K additionally required one documented admission-gate override).

2 RELATED WORK

LLM serving. KV-cache-centric serving (vLLM's PagedAttention (Kwon et al., 2023), FlexGen (Sheng et al., 2023), H2O (Zhang et al., 2023), StreamLLM (Xiao et al., 2024), Mooncake (Qin et al., 2025), SGLang's RadixAttention (Zheng et al., 2024)) optimizes allocation efficiency under high-bandwidth interconnects. Datacenter parallelism (Megatron-LM (Shoeybi et al., 2019), GPipe (Huang et al., 2019), PipeDream (Narayanan et al., 2019); prefill/decode disaggregation: Sarathi (Agrawal et al., 2023), Splitwise (Patel et al., 2024), DistServe (Zhong et al., 2024), DynaServe (Ruan et al., 2025)) scales serving across fast fabrics; offloading systems (DeepSpeed-Inference (Aminabadi et al., 2022), ZeRO-Infinity (Rajbhandari et al., 2021), HeadInfer (Luo et al., 2025)) trade slow-tier capacity for reach.

KV placement and transport decisions. Recent long-context systems decide *online where KV state lives and when it moves*—inside one accelerator's memory hierarchy or across a cluster fabric, never across the PCIe of a multi-GPU machine: ShadowKV (Sun et al., 2025) selects sparse KV pairs per decode step; TTKV (Dzikanyanga et al., 2026) assigns KV blocks to temporal tiers by recency; Tutti (Qiu et al., 2026) moves prefix KV between HBM and NVMe; Strata (Xie et al., 2025) caches hierarchical contexts across HBM/DRAM/SSD; TokenLake (Wu et al., 2025) places prefix segments across a cluster pool; Kareto (Zheng et al., 2026) configures tier capacities from a simulation-derived Pareto front. All optimize *within* one GPU+host hierarchy or *across* a fabric; none decides, per request, *which PCIe-connected accelerator* should hold KV over a narrow interconnect.

Multi-GPU KV transport and restoration over slow links. OrbitFlow (Ma et al., 2026) reconfigures per-layer KV placement with an ILP plus EWMA feedback to meet TBT/TPOT SLOs; Mell (Liu et al., 2025a) migrates KV states versus tokens across GPU pairs; CacheFlow (Nian et al., 2026) overlaps KV restoration with recompute; KVPR (Jiang et al., 2024) recomputes partial KV when I/O

is the bottleneck; Spin (Zhao et al., 2026), SparseServe (Zhou et al., 2025), HieraSparse (Wang & Wang, 2026) ship only selected KV under sparse attention. These decide *where KV lives and how it moves*; none decides, per request, *whether a second PCIe-connected accelerator should be used at all and in which mode*—our feasibility gate and mode selector. The layers are complementary: OrbitFlow-style placement is a policy *inside* a chosen mode; Seesaw (Su et al., 2025) re-shards prefill/decode within a mode; CacheFlow/KVPR/Spin-style techniques enter as transport-policy options that lower τ and shift $\mathrm{ctx}^*(B)$ rightward (Section 6.5); Appendix 23 measures one such plug-in end-to-end.

Serving over slow links. Infinite-LLM (Lin et al., 2024) splits attention along the sequence dimension so decode traffic over links as slow as 125 MB/s stays KB-scale; Helix (Mei et al., 2025) formulates heterogeneous-GPU serving as max-flow with MILP placement (ASPLOS 2025). Both assume datacenter fabrics and offline planning; we decide the same question online and per request. Closest in spirit: Harvest (Gopal & Kaffes, 2026) opportunistically caches peer HBM over NVLink with a when-to-harvest policy—the NVLink analog of our feasibility gate; ASTRA (Liu et al., 2025b) accelerates multi-device transformer inference under constrained interconnects, an option our cost model could absorb. Flying Serving (Gao et al., 2026) switches DP↔TP on the fly, but its premise—fine-grained grain-switching pays on fast fabrics—does not transfer to links too narrow for TP: our per-request mode decision is the coarse first question, grain switching a refinement inside a chosen mode. HILOS (Jang et al., 2025) and TraCT (Yoon et al., 2025) revise a topology's effective τ and ΔA at the substrate.

Scheduling and online learning. Topology-aware transfer scheduling is established in distributed training (ByteScheduler (Peng et al., 2019), TicTac (Hashemi et al., 2019)); learning-augmented scheduling (Lykouris & Vassilvitskii, 2018; Purohit et al., 2018) shares our predictor-plus-decision-rule paradigm; AIConfigurator (Xu et al., 2026) searches engine-level configurations (parallelism, batch, KV budgets) with a learned surrogate—our layer makes the per-request mode decision over a labeled topology, levels that compose. ThunderServe (Jiang et al., 2025) optimizes deployment plans under heterogeneous bandwidth; Vidur (Agrawal et al., 2024) simulates workload-conditioned serving and searches configurations offline—both search *within* a fixed topology, entering as an offline planner above our per-request mode decision.

Positioning. Table S50 (Appendix 22) places the systems we position against side by side on the τ interface: the KV-placement/selection and sequence-reformulation families are τ-reducers composing under our layer; fast-fabric parallelism and Harvest are fast-link-only; APEX is an alternative to a second GPU, not a reducer; we claim the per-request coarse mode decision—direct head-to-heads with these KV systems are not claimed.

CPU-assisted decode (APEX). A *single*-device alternative is a fourth mode, not a τ-reducer: CPU overlap (APEX (Fan et al., 2025)) moves decode work to the CPU. Appendix 22 brackets the boundary via the $\mathrm{ctx}^*(B)$ map and outlines the (unimplemented) fourth-mode absorption.

3 Problem Formulation

We model the hardware as a directed graph $G = (V, E)$ with per-edge bandwidth B_e and latency L_e, per-device usable memory M_v and compute throughput F_v; a path's effective bandwidth is its minimum edge bandwidth (notation in Table S1, supplement). The workload is $w = (N, C, R, \phi, q)$ (model size, context, concurrency, prefill/decode mix, quality target); the policy π covers residency, KV policy, transport, recompute, scheduling; we evaluate the triple (τ, w, π), not hardware in isolation. A multi-device configuration is *effective* if some policy yields a stable positive gain in at least one of four dimensions—feasibility, capacity, performance, and stability—formalized against the strongest single-device baseline by the stacking-effectiveness score $SE(\tau, w, \pi) = \max_d \gamma_d(\tau, w, \pi) - \lambda V_{\text{visible}}(\tau, w, \pi)$, where $\gamma_d \geq 0$ is the gain in dimension d (feasibility, capacity, performance, stability) and $\lambda \geq 0$ weights link-facing work; V_{visible} is negligible on NVLink-class links and dominant on narrow PCIe without peer access (Appendix 10 maps the five-tuple w onto the one-dimensional score used in deployment).

The three execution modes are defined physically. **Single**: one device holds the entire (quantized) model and all KV; nothing crosses the interconnect. **Staged-split**: layers are partitioned across devices; activations hand off at each boundary and decode relays the remote KV on every generated

token, so visible transport grows with context. **Replica (request-parallel)**: each device holds a full model copy and serves different requests; steady-state cross-device traffic is near zero, but per-card KV capacity constrains admissible context. Mode labels are physical: a plan labeled `single` that internally runs a split is scored as that split (Section 6.4). At runtime, each round chooses (m, π) from (τ, w), observes V_{visible} and γ_d, and updates π toward $\pi^* = \arg\min_\pi \sum_t T_{\text{step}}(\tau, w_t, \pi)$. To bound visible cost on any topology, we decompose per-step latency as

$$T_{\text{step}} = \max(T_{\text{compute}}, T^{\text{visible}}_{\text{transport}}, T_{\text{kv-miss}}, T_{\text{sync}}), \tag{1}$$

yielding three derived properties—communication hiding, throughput scaling, recovery control (Appendix 10).

4 Method: The Topology-Aware Decision Framework

The framework has four control primitives over G and a five-tier state taxonomy (hot/ tail/ compressed/ host RAM/ disk; Figure 2, appendix): **Place** maps state objects under per-device budgets (entropy-based assignment prefers high-entropy layers near the input); **Move** schedules transfers before their consumer needs them; **Reuse** restores prefix KV from the warm tier to skip prefill; **Recompute** treats rebuilding state as a normal cost decision. Each request runs a fixed loop: restore cached prefix; prefill through the stages; decode with on-demand relay; cache the prefix KV; evict LRU to the warm tier (Algorithms S1–S2, appendix).

The refit selector. The decision layer's mode score is predicted per-request latency under

$$T_{\text{split}} = A_S + b_S\,\text{ctx} + c_S\,\text{ctx}^2, \qquad T_{\text{replica}} = A_R + b_R\,\text{ctx} + c_R\,\text{ctx}^2, \tag{2}$$

with per-mode hard memory feasibility as the eligibility mask (a per-model replica KV slope and a concurrency-saturating staged activation term, Section 6.4), a measured batch penalty for replicas at concurrency ≥ 2. The fit is deliberately low-dimensional (three coefficients per mode; RMS 1,383 ms); leave-one-context-out refits hold held-out latencies to 2.7 s RMS, and ΔA trades off against curvature, so the crossover rests on measured grids (Appendix 5.2). The latency score is negated into a reward ($S_m = -T_m$) so that greedy, UCB, and Thompson all take an argmax; equivalently, all three choose the feasible mode with the lowest predicted latency. UCB and Thompson are compared (Auer et al., 2002; Agrawal & Goyal, 2012; Russo et al., 2018) (appendix).

Engine integration and unified decision protocol. The decision layer is engine-agnostic: it emits *configurations* for an execution engine (Section 6.8), deciding sizes and modes without touching the engine's KV ownership. Per request: (1) the *feasibility gate* (per-mode predicted peak vs. the 95% usable budget, Section 6.4) narrows the mode set; (2) the EWMA-calibrated *latency score* (Eq. (2)) ranks feasible modes; (3) inside the decision noise band ($|T_{\text{split}} - T_{\text{replica}}| \leq 2\epsilon_t$) the *measured regime law* decides instead of the fit (replica at conc-1, split at conc-$\geq$2, single-card when fastest).

Calibration at a glance. Coefficients are calibrated per session from a dual-mode latency ladder (short/mid/long contexts; n=3–5 warm reps per cell); online only τ moves, via the *held-fit* $\tau_{\text{held}} = \text{mean}_{\text{ctx}}[(D(\text{ctx}) + \Delta A - \Delta c\,\text{ctx}^2)/\text{ctx}]$ over probe contexts (ΔA, Δc fixed at the ladder), with per-request prediction errors folded into an EWMA bias correction (α=0.9: stationary scatter $0.229\,\sigma$, i.e. a 19-observation average; error halves in $\sim$7 rounds, $< 5\%$ in 29; Appendix 2). The decision noise band is $2\epsilon_t$—twice the fit RMS (ϵ_t=1,383 ms; $\approx$2.8 s)—inside which the selector follows the measured regime law, not the fit. A five-context bootstrap probe recovers τ within $+0.3\%$ bias, $\pm 11\%$ (1σ), at a 0.2% flip rate (even k=2: $< 2\%$ bias, $\leq 0.4\%$ flips, $\sim$10 min; Appendix 16.1); a 2σ alarm triggers the 5-context refresh, and the full ladder is re-fit only on alarm or topology change. Decisions tolerate $\pm 15\%$ error on either coefficient (first flip at $\approx 31\%$; Appendix 5.3), stabilize in 100–160 observations under regime shift (Appendix 16); four fresh sessions recalibrated with zero flips (32/32; Appendix 19).

5 Analysis

The framework follows the predictor-plus-decision-rule architecture: the cost model predicts, the mode selector decides, and the results below relate decision quality to prediction error. We state six supporting results; the tools are standard, the contribution is their instantiation—each guarantee is tied to quantities measured on the target platform (proofs in Appendix 10).

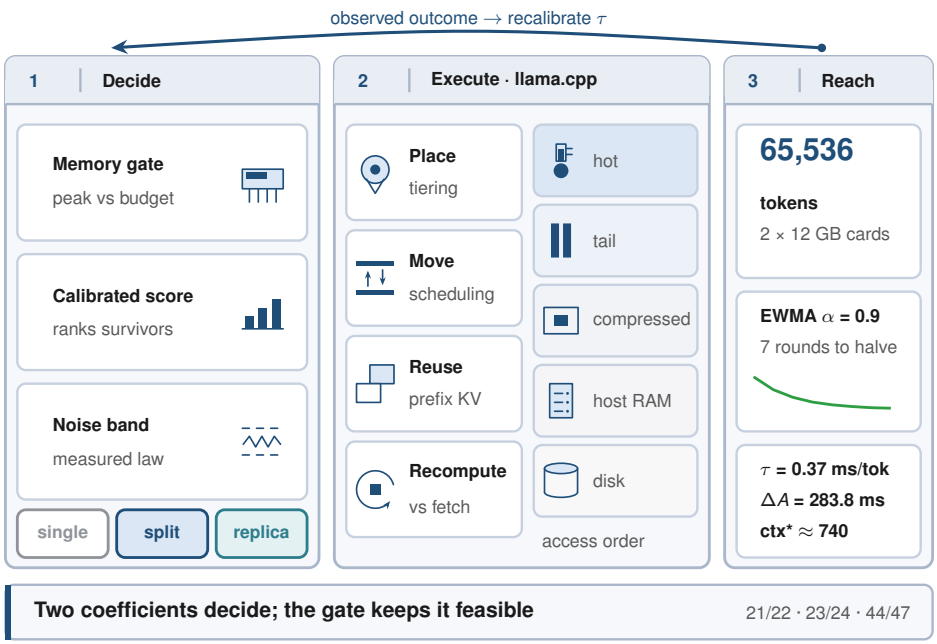


Figure 2: Our decision layer narrows per-request mode candidates by a memory-feasibility gate and ranks the survivors with a calibrated latency model, emitting an engine configuration whose observed outcomes are folded back (EWMA) — so the narrow-interconnect system commits per request to single / staged-split / replica. Per request the runtime restores the cached prefix, prefills and decodes across the PCIe-4.0-x8 relay (no NVLink; $B_e \approx 13.3$ GB/s), then caches the prefix and evicts LRU (recompute on prefill failure); state primitives (Place/Move/Reuse/Recompute) run over the five-tier taxonomy (Algorithm S1).

(i) **Stack-roofline decomposition** of per-step latency (Eq. (1)); (ii) **Prediction-error propagation bound**: the greedy selector's per-step suboptimality is at most $2\epsilon_t$, twice the cost model's transport-estimation error, so its cumulative regret is bounded by $2\sum_t \epsilon_t$—instantiated empirically in Section 6.4 (RMS 1,383 ms). (iii) **Topology-sensitive scaling law**: the critical device count beyond which stacked throughput saturates ($\approx$477 for the boundary-handoff payload, $\approx$451 substituting the measured τ; Appendix 12). (iv) **Mode-crossover criterion** (Section 6.5): under Eq. (2), $D(\text{ctx})$ is strictly increasing with exactly one positive root $\text{ctx}^* \approx \Delta A/\tau$; since τ scales inversely with bandwidth, $\text{ctx}^*(B)$ grows linearly while the replica's memory ceiling is bandwidth-independent—the U-shape re-emerges on fast links. (v) **EWMA convergence** (geometric bias decay); (vi) **Sublinear regret** for UCB/Thompson (standard); Section 6.6 shows the practical lever is estimator memory, not exploration.

6 Evaluation

6.1 Experimental Platform: A Representative Consumer Topology

The platform is a consumer workstation—one concrete narrow-interconnect instance—with **two NVIDIA RTX 3060 12 GB GPUs** on PCIe 4.0 x8 (no NVLink, no P2P; all cross-GPU traffic traverses the CPU relay—the x8-shared-root case), an **AMD Ryzen 9 9950X3D** host with **16 GB DDR5 RAM** (Windows 11, PyTorch 2.11, Transformers 5.11.0), and Qwen/Qwen2.5-7B-Instruct (28 layers, 7B parameters); its topology is a CPU-relay star ($B_e \approx 13.3$ GB/s measured). Baselines: **Single-GPU** (quantized), **HF Auto**, **CPU Offload**, **Naive Pipeline**, **Our Runtime**. All matrices use tokenizer-verified true-token prompts, distinct prompts per repetition, uniform max_new_tokens $= 64$; warm latencies exclude process-cold and first-at-length taxes (appendix). vLLM runs under WSL2 (no native Windows build for this card class); the fairness-controlled pairing is Section 6.8.

Single-GPU execution OOMs at load in bf16 at all lengths (quantized probe: 3,072-token boundary). From 1,024 tokens the staged runtime beats the strongest baseline (-7%, -24%, -27% at 1,024/2,048/3,072), significant at 2,048 (Welch $t = -4.03$, df$= 3.0$, $p = 0.028$; remaining cells underpowered at $n = 2$). The mechanism is entropy-aware placement: 14 high-entropy layers per GPU with a *single* cross-GPU boundary, against *ten* boundaries for HF Auto's automatic device map (Appendix 24; Figure S3(b)), score 7–27% faster from 1,024 tokens with symmetric peaks ($\sim$7.4 vs. up to $9.0/10.8$ GB/GPU); plan-time profiling costs one 0.78 s forward pass (appendix).

6.2 Verified Long-Context Matrix

The verified long-context matrix (Appendix 18): every tested context through 65,536 true tokens completes (twice the native window, dynamic NTK; 6,652 MiB binding peak), while HF `device_map="auto"` OOMs at 8,192 and llama.cpp refuses admission past 32K; latency grows 13.2 s (8K) to 180.0 s (65K). This is the *reach-and-automation* result; retrieval quality is measured separately below.

Memory accounting at 65,536 tokens. The "65K bf16 KV needs tens of GiB" assumption holds only for MHA (28.0 GiB); both served models use GQA, so full-precision KV at 65K is 3.50/8.00 GiB, and in 4-bit staged mode the KV splits across cards to measured peaks (Qwen 4,419/6,652 MiB; Llama 6,960/9,095 MiB) inside the 12-GiB envelope; host DRAM carries the warm tier (4 GiB cap).

Prefix reuse: end-to-end speedup grows with prefix length (1.7% at 128 tokens to 49.4% at 4,096; pooled $p = 0.0023$), and a 512-token ablation isolates reuse as the only component measurable at that length; quality at the frontier (Appendix 18): 42/42 fixed-design and 30/30 randomized-needle retrievals at 65K, a NoLiMa sweep matching the 7–8B pattern, and SCBench on its official metrics—choice accuracy $0.600/0.588/0.650$ and QA accuracy $0.110/0.130/0.210$ at 32K/65K/128K (ExLlamaV3 4.0bpw; single GPU at 32K/65K, tensor-parallel at 128K). Choice sits at or above SCBench's published bf16 8–9B band but QA below it (0.29–0.34): the QA shortfall is real; the claim is the narrower one—no degradation that *grows* with context to 128K.

6.3 Mainstream-Engine Baselines: llama.cpp, ExLlamaV3, and vLLM

Against the strongest mainstream consumer engines, same true-token convention: llama.cpp (ggml-org, 2026) (b10066, Q4_K_M), ExLlamaV3 (turboderp-org, 2025) (4.0bpw, TP), vLLM (Kwon et al., 2023) (0.25.1, AWQ, PP/TP, WSL2); Table S38, Appendix 18 gives the ladder. Three *decision-layer capability* findings (Appendix 27). **Speed at $\leq$32K**: llama.cpp's Q4_K_M kernels are $\approx 3.7\times$ faster than our on-the-fly-NF4 staged path, ExLlamaV3's TP 1.2–$1.7\times$ *slower*, vLLM (AWQ, PP-2) 1.6–$2.3\times$ faster—our HF vehicle loses to optimized kernels. **Reach**: llama-server clamps admission to its GGUF window (32,768; slot clamp 32,640; planner-driven llama.cpp measured only to 32,000, and none lifts either on b10066 (llama.cpp project, 2025)), while our 4-bit staged path enables dynamic NTK and reaches 65,536. That reach is *not* exclusive—vLLM admits 65K via a hand-set `max_model_len`/RoPE override (vLLM Project, 2025), and ExLlamaV3 with stock RoPE; both are faster there (44.2 and 136–164 s vs. our 180.0 s). **Automation**: the framework quantizes, admits, and extends context by planner decision, with no manual quantization, cache sizing, or RoPE tuning—at the frontier the differentiation is automation, not speed: static defection pays $+13\%$–$+87\%$ (Appendix 27.1).

Matched-stack isolation and native-Linux parity. Matched native-Linux runs (same GGUF, same prompts; Appendix 27.1) isolate the decision layer from kernel, quantization, and OS effects and reproduce every verdict under a single stack; the layer's increment is Section 6.8. Under those matched conditions the engine-axis crossover is engine-dependent (llama.cpp $-38\%/-35\%$, SGLang $-38\%/-40\%$, vLLM TP2 parity at 16K only; Appendix 27)—the per-engine τ the decision layer must recalibrate.

6.4 Selector Refit: Repairing the Three Diagnosed Mechanisms

A learned selector was the least reliable component—only 45.0% correct against measured hindsight (9/20 grid cells; 55% wrong), the errors systematic, not boundary jitter. Diagnosis traced them

to three mechanisms, each repairable with measured calibration: execution-label mismatch, wrong-sign concurrency credit, and missing per-mode feasibility terms. The repaired score becomes predicted per-request latency under Eq. (2) (Appendix 5), recovering $\tau = b_S - b_R = 0.37$ ms/token, $\Delta A = 284$ ms. Re-scored: **21/22** on the grid and **23/24** across all 24 points at n=5 warm reps (feasibility 6/6, crossover 7/8, stability 6/6, long-context 2/2), 4/4 fresh refit cells, and 44/47 on three off-grid sets (Appendix 5); the single miss (2,048×c1) is a fourth mechanism—true single-card execution beats both two-GPU modes at small context; observed suboptimality is 0 ms across all 24 points. On the deduplicated off-grid set the calibrated layer wins 33/36 vs. 23/36 against a *correctly implemented one-line static rule* (Appendix 5), its increment being short-context splits and the feasibility gate's OOM rejections.

6.5 The Mode-Crossover Condition and Its Validity Region

The condition. With both modes feasible, $D(\text{ctx}) = T_{\text{split}} - T_{\text{replica}} = -\Delta A + \tau\,\text{ctx} + \Delta c\,\text{ctx}^2$. The derivation from Eq. (2) is elementary; what is *measured* is portability—after per-engine recalibration: the coefficients, their scales, and where it stops holding. The monotonicity condition $D'(\text{ctx}) = \tau + 2\Delta c\,\text{ctx} > 0$ holds whenever $\tau > 0$ and $\Delta c \geq 0$, giving exactly one positive root $\text{ctx}^* \approx \Delta A/\tau$; the curvature term shifts the root by $< 8\%$ (740 vs. 769 tokens; Appendix 5.2)—yet the exact root is the fit's least-identified parameter (LOCO spread 120–1,281; ΔA can flip sign; Appendix 5.2); what is robust is the discrete decision, which $\pm 15\%$ error flips not (first flip $\approx 31\%$). *We observed $\Delta c > 0$ on the single platform where it is identifiable*. Table 1 gives the coefficients across engines and hosts ($\tau = b_S - b_R = 0.37$ ms/tok, $\Delta A = 284$ ms).

Table 1: The two coefficients and the crossover they imply, as measured across engines and hosts (ΔA in ms, Δc in ms/tok^2; protocols in Appendix 10.2). "No root" = $\Delta A < 0$, the replica wins at every context.

Engine/host	τ (ms/tok)	ΔA (ms)	Δc	ctx*
HF vehicle, x8 3060s, Qwen-7B 4-bit (fitted / held-fit)	0.37 / 0.56	284	$+1.9\times10^{-5}$	769; 507 (274 Gen3)
llama.cpp, x8, Qwen-7B Q4_K_M	$\ll 0.37$	—	—	$\leq$ 1,024 *(measured)*
Framework, 4×4090 x16 (B=25.4 GB/s), Qwen-7B / 14B	0.048 / 0.125	$\Delta A < 0$	—	no root

With the SGLang tp2 and llama.cpp axes the engine-dependent crossover is *measured, not fitted*: SGLang's split beats its single arm by 38–40% at 8K/16K while vLLM's TP2 splits lose at 8K (+12%/+30%), so per-engine calibration is required (Algorithm S1, EWMA, Section 5)—and sufficient, since the calibrated selector still makes zero-cost decisions after per-session recalibration (32/32 shared contexts, Appendix 19). The linear link relation is validated by the BIOS Gen3 downgrade (B 13.3 → 6.65 GB/s raises τ 0.56 → 1.04 ms/tok, 1.85× vs. predicted 2.0; ctx* 507 → 274) and yields the linear-in-B shape of ctx$^*(B)$ (Figure 3).

Beyond k=2: measured, with a mechanism. Table 2 gives the complete k=1–4 picture on a no-NVLink 4×RTX 4090 (measured relay B=25.4 GB/s; n=8 worker reps / n=5 kway reps, 1σ; Appendix 20.1). *Replica is near-linear through k=4*: independent single-card workers reach $\times 1.98/2.96/3.89$ (7B) and $\times 1.98/2.98/3.89$ (14B), i.e. $\times k/k$=0.97–0.99 (3.96–4.00 on a second host whose four links are uniform)—while the framework's batching dispatcher is $\times k$*-dependent* (host-sensitive, $\times 1.03$ short-context to $\times 2.75$ at 65K), so independent-worker scaling is the robust ceiling. *The k-way split tax is k-insensitive*: τ is flat through k=3 (τ_3=0.91/1.09 of τ_2) and τ_4/τ_2=0.74–1.26 across hosts/fits—far below the shared-root $(k-1)^2$=9 and per-pair $(k-1)$ forms, which these distinct-bridge hosts reject ($\tau_k \approx \tau/k$ fails too). The plausible physical account matches: on these hosts cards sit on distinct host bridges, so relays do not serialize at one root complex, and the tax is set by single-hop activation, which does not grow with hops. k=8 remains extrapolation.

The coefficients as an interface. Because τ is system-controllable, the crossover claim is also a composition claim: a transport-reducing technique lowers τ and shifts ctx* rightward *without any change to the decision layer*. We measure one such plug-in end-to-end: fp8 KV cache on SGLang (tp2, AWQ, same session) lowers the per-token slope 0.171 → 0.149 ms/tok (-13% E1-consistent; -24% steady) and shifts the crossover $+15\%$/$+31\%$ at fixed ΔA (-3 to -12% at ctx $\geq$ 8K; band-conditional). Two measured qualifiers bound the claim (Appendix 23). (i) A task-level needle probe shows this fp8_e5m2 instantiation is quality-failing (0/48 vs. 48/48 bf16; disjoint Wilson

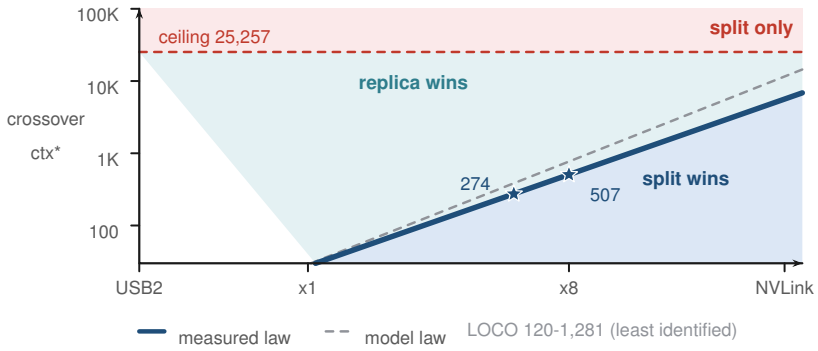


Figure 3: The mode crossover. $\mathrm{ctx}^*(B)$ (dashed, τ_0=0.37 ms/tok at B_0=13.3; solid: measured held-fit law) through the Gen4/Gen3 anchors ($\star$: halving B raises τ 0.56→1.04 ms/tok and lowers ctx^* 507→274); the replica's memory ceiling (dotted) is bandwidth-independent; x-axis annotated with physical link points.

Table 2: Measured k=1–4 on a no-NVLink 4×RTX 4090 (independent single-card workers for replica; k-way split for τ; n=8 worker reps, n=5 kway reps, 1σ; τ_k is the split arm's own slope on a second, uniform-link host, not Table 1's $b_S - b_R$; full per-point data and second-host reproductions in Appendix 20.1). Both the shared-root $(k-1)^2$ and the per-pair $(k-1)$ predictions are falsified on distinct-bridge hosts; shared-root $k>2$ remains unmeasured.

Model	×2	×3	×4	$\tau_2/\tau_3/\tau_4$ (ms/tok)
7B	1.977±0.024	2.958±0.097	3.894±0.095	0.051/0.046/0.064
14B	1.980±0.037	2.982±0.075	3.889±0.091	0.111/0.121/0.118

intervals): the plugin demonstrates the *mechanism* but is not a deployment-recommended τ-reducer. (ii) A relay-side byte probe on the framework's own k=2 split (Appendix 24) shows its per-token tax is sync-bound rather than byte-bound, and in TP serving the fp8 effect is on attention-memory reads, not relay bytes—so the byte-linear r-scaling applies to byte-bound axes (the SGLang axis above), while the framework axis is disclosed as the sync-bound case. This is the mechanism by which OrbitFlow/ShadowKV/TTKV-style systems compose with our layer, with their own quality cost measured before adoption.

Where the crossover breaks. The single-root guarantee requires $\Delta A > 0$ and $\Delta c > 0$. On the fastest host (4×RTX 4090, measured relay B=25.4 GB/s, τ≈0.05 ms/tok) ΔA<0: the crossover vanishes and the replica always wins—the fast-link limit. Protocol dependence (bracket $(512, 1536–2048)$) and per-session drift (held-fit τ +52–215%, no flips) bound trustworthiness (appendices).

6.6 Bandit Selector Simulation Study

We simulate $T = 500$ rounds over 20 seeds on score distributions calibrated on measured configurations (greedy, UCB, Thompson, recency-weighted; (Garivier & Moulines, 2011)). Under stationarity the bandit selectors are *worse* than greedy (+113%; $t \approx$ 11–12): exploration costs, not pays. The dominant error under regime shifts is *estimator staleness*, not exploration; sliding-window greedy ($W = 100$) cuts regret 17.1% ($t = -3.76$, $p < 0.01$).

6.7 Cross-Model Validation: Llama-3-8B-Instruct

The same verified protocol runs on Llama-3-8B-Instruct (GQA, 8 KV heads) with zero code changes: every context through 65,536 true tokens completes in 4-bit staged mode. One gate asymmetry is disclosed: the budget gate rejects Llama at 65,536 at load time (conservative; measured

peaks 6,960/9,095 MiB sit 2.9 GiB inside the 12-GiB envelope on the binding card, so 65K is measured with the gate overridden); the corpus audit (Appendix 13.2) records zero false positives on the 24-cell grid (32 corpus-wide, none on the grid) and this is the gate's only conservative false negative, safe at the measured headroom. Llama's $2.29\times$ KV penalty shows in the memory ceiling, not per-call latency, and the per-model KV-slope factor does not transfer ($3.424\times$ Qwen2.5-7B vs. $1.91\times$ Llama-3-8B, both measured against the bf16-KV slope), so the selector learns it per model (EWMA). **Transfer across hosts and model sizes.** The same protocol runs on a second host (x16, 17.6 GB/s, two RTX 4090 24 GB) for Qwen-7B/14B: the replica wins every feasible conc-1 context (49/49; margins $+5.7\%$–$+93\%$), with ceilings following the gate's KV-ratio arithmetic (Appendix 13). **Architecture axes.** Two further axes (Appendix 31): an MHA model (Llama-2-7B, $9.14\times$ KV) and a MoE model (Qwen1.5-MoE-A2.7B, $3.43\times$ KV) both land in the replica-dominant regime (crossover $< 256/< 512$; fitted $\Delta A{<}0$), and the gate's KV arithmetic predicts the measured MHA replica ceiling $(8{,}192, 16{,}384)$.

6.8 Planner-Driven Engine Integration: llama.cpp

The decision layer integrates above llama.cpp kernels: the refit selector and admission gate emit llama.cpp configurations. **Automation on fast kernels**: planner-driven runs execute llama.cpp's kernels, so the $\approx 3.7\times$ deficit does not apply (8.37 vs. 6.44 s at 8,192: the planner's replica loses 23% to the hand-tuned split; Appendix 18). **The mode crossover is engine-dependent**: llama.cpp's split carries a far smaller transport tax—its crossover is at or below $1{,}024$, against our < 512—so the HF-calibrated decision diverges from $4{,}096$ (split wins 13–25%): EWMA-absorbed per-engine calibration (barriers: admission clamp, KV-layout ownership; appendix). **The decision layer's own increment**: under the fully matched native-Linux stack (Section 6.3) the calibrated split is the measured best mode throughout, and each static defection pays $+13\%$–$+87\%$ (Appendix 27.1).

6.9 Evaluation Breadth: Decode Lengths, Concurrency 8, Contention

Evaluation breadth. Forced full generations at 8,192 ($n{=}4$ warm): the replica wins every decode length ($-19.4/-16.4/-12.6\%$ at 128/256/512); at concurrency 8 the staged path wins both fresh cells (-4.5% at 4,096×8; replica OOMs at 8,192×8), falsifying the linear activation term; the c4 verdict is stack-dependent (staged on Windows, replica on WSL2; EWMA-absorbed). The batch penalty enters the same abstraction as a coefficient shift: it falls on the replica, raising that arm's per-token slope, so the tax $\tau = b_S - b_R$ shrinks and the crossover $\mathrm{ctx}^* = \Delta A/\tau$ *moves right*—the split's win region widens with pressure, which is exactly the measured regime law (replica at c1, split at c$\geq$2; framework $+39$–59%, llama.cpp $+9.5$–87% at $c{=}2$, Appendix 28). Contention robustness (rate-limited sweep 0–22 GB/s: every verdict holds, replica 24/24, margins $+29.9\%$–$+67.5\%$; Table S37–Appendix 15) holds: the split slows $+9\%/+3\%$, the replica $+19\%/+19\%$. **Decode-length attribution.** Mode choices are prefill-τ dominated: the split-vs-replica gap rises only ≈ 4 ms/tok over $G{=}128$–$1{,}024$ (Appendix 31).

7 Discussion and Limitations

Limitations. The evaluation spans five models (two headline) and five engines, four link settings on three hosts, and a 12–24 GB card family; no $>$14B model, NVLink-class link, or shared-root $k{>}2$ host is measured. The grid is finite (44/47 exact), the crossover is bracketed, and the dispatcher's $\times k$ is host-dependent.

8 Conclusion

The decision layer answers both halves of the question: memory feasibility decides *whether* a second accelerator helps, and the calibrated crossover $\mathrm{ctx}^* = \Delta A/\tau$ decides *when*. Because τ is system-controllable, the same two coefficients act as an interface—transport-reducing techniques shift the boundary without changing the decision layer. A bandwidth-starved topology is therefore a schedulable resource rather than a fixed limit: 65,536 tokens on two 12 GB consumer cards, with no manual quantization or RoPE tuning.

AI USE STATEMENT

In this work, we used generative AI tools (large language models) for: drafting and editing portions of the manuscript (copyediting, readability, LaTeX troubleshooting); summarizing and analyzing the literature for the related-work discussion; brainstorming and refining the paper's narrative, hypotheses, and positioning (including the two-coefficient interface framing and the title); formulating and deriving the paper's mathematical claims and the appendix proofs (standard UCB/TS regret bounds and EWMA convergence results, instantiated and checked by the authors); designing and refining scientific figures; creating and editing portions of the measurement and analysis code (all code was executed on real hardware and verified by the authors before use); providing feedback on the experimental methodology (e.g., suggesting additional measurement grids, statistical checks, and robustness probes); and supporting the analysis and interpretation of measured results. We did not use generative AI tools for: generating synthetic datasets; executing experiments or collecting measurements (every reported number comes from a real hardware run of the released code); or cleaning or reformatting datasets (measurement artifacts are raw per-repetition JSON files). Translation and qualitative/thematic data analysis are not applicable to this work. We have reviewed all AI-assisted work: an automated audit script re-derives the quantitative claims against the released artifacts (84 claim nodes: 67 exact passes, 16 caliber-caveated, and one legacy counterfactual reconstructed from the enumerated grid); AI-assisted text was checked for plagiarism through a manual literature survey; AI-assisted code was tested on the target hardware; and AI-assisted figures were audited for consistency with the source data. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI. For completeness: no generative model was used to interpret the raw model outputs behind the retrieval-quality measurements, which the released evaluation scripts score; no figure is produced by a generative model at the image level, since every panel comes from released generators reading committed data files; and where a model proposed an analysis, the decision to run it and the resulting numbers come from the released code. The claim audit and the figure generators ship with the bundle, so the checks described here can be re-run end to end by a reviewer.

ETHICS STATEMENT

This work studies LLM serving efficiency on consumer multi-GPU hardware; it involves no human subjects, no personal or sensitive data, and no dual-use artifacts beyond standard open-source models, runtimes, and benchmark protocols that are publicly documented. All measurement artifacts and the anonymized code bundle are released for reproducibility; no new datasets are introduced. We have read and adhere to the ICLR Code of Ethics. No user data, no personally identifiable information, and no third-party model outputs were collected or released; all models and benchmarks are public artifacts obtained through their official distributions. The framework lowers the hardware barrier to long-context inference on consumer GPUs; like any serving-efficiency result it is dual-use in principle, but it adds no model capability, no fine-tuning, and no content-generation method, only a more efficient schedule for existing inference under a bandwidth constraint. Model weights are not redistributed: the bundle carries only code, protocols, and measurement artifacts.

REPRODUCIBILITY STATEMENT

The released code bundle (`code.zip`) reproduces every measurement in Section 6 from committed scripts and per-repetition JSON artifacts: selector calibration and re-scoring (including leave-one-context-out refits), the verified long-context matrix, the RULER-style needle probes, the contention and concurrency suites, the cross-OS/SGLang/arrival studies, and the planner-driven llama.cpp integration. Full protocols (true-token construction, nonce design, tax definitions) are in the appendix that follows the references. The bundle also records the platform (two RTX 3060 12 GB on PCIe 4.0 x8 with no NVLink; a host with two RTX 4090 24 GB on PCIe 4.0 x16 across NUMA nodes; a four-card no-NVLink host with a 25.35 GB/s relay) and pins the software versions (PyTorch 2.11, Transformers 5.11.0, llama.cpp b10066, ExLlamaV3 1.1.0, vLLM 0.25.1). Warm-up conventions, the raw per-repetition JSON artifacts, and the aggregation scripts are included, so every reported mean, standard deviation, and significance test can be recomputed without re-running GPU work. Named entry points cover the selector evaluation, latency calibration, the crossover-law check, the leave-one-context-out refit, SCBench, the cost-model cells, the needle probes, and the llama.cpp planner, plus unit tests.

REFERENCES

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. SARATHI: Efficient LLM inference by piggybacking decodes with chunked prefills, 2023.

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for LLM inference. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 6, pp. 351–366, 2024.

Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Proceedings of the 25th Annual Conference on Learning Theory (COLT)*, pp. 39.1–39.26, 2012.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. DeepSpeed inference: Enabling efficient inference of Transformer models at unprecedented scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022.

Peter Auer and Ronald Ortner. UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61(1–2):55–65, 2010. doi: 10.1007/s10998-010-3055-6.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002.

Tian Chen, Mingheng Mi, Pu Wang, Chen Hu, and Hai Zhang. Automatic model-hardware co-adaptation for heterogeneous AI accelerators, 2026. Presents MetaInfer, an LLM-agent system that formulates inference adaptation as route search over a costed execution-adaptation graph, patching adapters, runtime dispatch, kernels and deployment configuration; evaluated on NVIDIA A800 and Hygon K100AI DCU.

Gradwell Dzikanyanga, Weihao Yang, Hao Huang, Donglei Wu, Shihao Wang, Wen Xia, and Sanjeeb KC. Ttkv: Temporal-tiered kv cache for long-context llm inference, 2026.

Jiakun Fan, Yanglin Zhang, Xiangchen Li, and Dimitrios S. Nikolopoulos. APEX: Asynchronous parallel CPU-GPU execution for online LLM inference on constrained GPUs, 2025.

Shouwei Gao, Junqi Yin, Feiyi Wang, and Wenqian Dong. FLYING SERVING: On-the-fly parallelism switching for large language model serving. In *Proceedings of the 40th ACM International Conference on Supercomputing*, pp. 17–29, 2026. doi: 10.1145/3797905.3800525.

Aurélien Garivier and Éric Moulines. On upper-confidence bound policies for switching bandit problems. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT)*, pp. 174–188, 2011.

ggml-org. llama.cpp: LLM inference in pure C/C++. https://github.com/ggml-org/llama.cpp, 2026. Build b10066-86a9c79f8 used as the optimized-kernel baseline throughout; accessed 2026-07-19.

Vikranth Gopal and Kostis Kaffes. Harvest: Opportunistic peer-to-peer GPU caching for LLM inference, 2026.

Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H. Campbell. TicTac: Accelerating distributed deep learning with communication scheduling. In *Proceedings of Machine Learning and Systems (MLSys)*, pp. 418–430, 2019.

Tingbo He. A time scaling theory for multi-layer electronic systems. *Science China Information Sciences*, 69(8), 2026. doi: 10.1007/s11432-026-5028-5. Journal version DOI 10.1007/s11432-026-5028-5 (verified via Crossref); earlier preprint: ChinaXiv 202605.00224.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Reiman, Feiyang Jia, and Boris Zhang. Ruler: What's the real context size of your long-context language models?, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Hongsun Jang, Jaeyong Song, Changmin Shin, Si Ung Noh, Jaewon Jung, Jisung Park, and Jinho Lee. A cost-effective near-storage processing solution for offline inference of long-context llms, 2025.

Chaoyi Jiang, Lei Gao, Hossein Entezari Zarch, and Murali Annavaram. KVPR: Efficient LLM inference with I/O-aware KV cache partial recomputation, 2024.

Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, and Eiko Yoneki. ThunderServe: High-performance and cost-efficient LLM serving in cloud environments. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 7, 2025.

Gregory Kamradt. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for Large Language Model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. ACM, 2023.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi: 10.1017/9781108571401. See in particular Chapter 7, "The Upper Confidence Bound Algorithm".

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods, 2024.

Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, Shen Li, Zhigang Ji, Tao Xie, Yong Li, and Wei Lin. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache, 2024.

Qianli Liu, Zicong Hong, Fahao Chen, Peng Li, and Song Guo. Mell: Memory-efficient large language model serving via multi-gpu kv cache management, 2025a.

Xiao Liu, Lijun Zhang, Deepak Ganesan, and Hui Guan. ASTRA: Communication-efficient acceleration for multi-device transformer inference, 2025b.

llama.cpp project. llama.cpp issue #17459: Misc. bug: Can't use longer context than model via RoPE due to server-imposed restrictions. https://github.com/ggml-org/llama.cpp/issues/17459, 2025. Official llama-server hard-clamps slot context to the GGUF n_ctx_train window even under explicit YaRN flags (behavior traced to commit cd5e3b5); accessed 2026-07-19.

Cheng Luo, Zefan Cai, Hanshi Sun, Jinqi Xiao, Bo Yuan, Wen Xiao, Junjie Hu, Jiawei Zhao, Beidi Chen, and Anima Anandkumar. HeadInfer: Memory-efficient LLM inference by head-wise offloading, 2025.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 3296–3305, 2018.

Xinyue Ma, Heelim Hong, Taegeon Um, Jongseop Lee, Seoyeong Choy, Woo-Yeon Lee, and Myeongjae Jeon. OrbitFlow: Slo-aware long-context llm serving with fine-grained kv cache reconfiguration, 2026. Proceedings of the VLDB Endowment, vol. 19; DOI 10.14778/3796195.3796214.

Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving Large Language Models over heterogeneous GPUs and network via Max-Flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 1*, pp. 586–602, 2025. doi: 10.1145/3669940.3707215. Published version of the 2024 arXiv preprint.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. Nolima: Long-context evaluation beyond literal matching, 2025. ICML 2025.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Nikhil Seshadri, NR Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM, 2019.

Sean Nian, Jiahao Fang, Qilong Feng, Zhiyu Wu, and Fan Lai. CacheFlow: Efficient llm serving with 3d-parallel kv cache restoration, 2026.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pp. 118–132, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. OpenReview forum id `wHBfxhZu1u`.

Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 16–29, 2019.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pp. 9684–9693, 2018.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation – A KVCache-centric architecture for serving LLM chatbot. In *Proceedings of the 23rd USENIX Conference on File and Storage Technologies (FAST)*, pp. 155–170. USENIX, 2025.

Shi Qiu, Yifan Hu, Xintao Wang, Wenhao Zhu, Jianqin Yan, Hao Chen, Kaiqiang Xu, Kai Chen, and Yiming Zhang. Tutti: Making ssd-backed kv cache practical for long-context llm serving, 2026.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. ZeRO-Infinity: Breaking GPU memory wall for extreme scale deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2021.

Chaoyi Ruan, Yinhe Cai, Dongqi Tian, Yandong Shi, Yongji Wu, Jialin Li, and Cheng Li. DynaServe: Unified and elastic execution for dynamic disaggregated LLM serving, 2025.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. High-throughput generative inference of Large Language Models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*. PMLR, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism, 2019.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In *Proceedings of the 31st IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

Qidong Su, Wei Zhao, Xin Li, Muralidhar Andoorveedu, Chenhao Jiang, Zhanda Zhu, Kevin Song, Christina Giannoula, and Gennady Pekhimenko. Seesaw: High-throughput LLM inference via model re-sharding, 2025. MLSys 2025.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, pp. 57355–57373, 2025.

turboderp-org. ExLlamaV3: quantization and inference library for LLMs. https://github.com/turboderp-org/exllamav3, 2025. Version 1.1.0 precompiled wheel (cu128, torch 2.11.0); native tensor-parallel backend and official 4.0bpw EXL3 artifact used for the baseline; accessed 2026-07-19.

vLLM Project. vLLM engine arguments. https://docs.vllm.ai/en/latest/configuration/engine_args.html, 2025. Documents max_model_len admission validation (over-length prompts rejected with HTTP 400) and YaRN-based rope scaling as the manual route beyond the declared window; accessed 2026-07-18.

Haoxuan Wang and Chen Wang. HieraSparse: Hierarchical semi-structured sparse KV attention, 2026.

Bingyang Wu, Zili Zhang, Yinmin Zhong, Guanzhe Huang, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Tokenlake: A unified segment-level prefix cache pool for fine-grained elastic long-context llm serving, 2025.

Guangxuan Xiao, Yu Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical context caching for long context language model serving, 2025. OSDI 2026.

Tianhao Xu, Yiming Liu, Xianglong Lu, Yijia Zhao, Xuting Zhou, Aichen Feng, Yiyi Chen, Yi Shen, Qin Zhou, Xumeng Chen, Ilya Sherstyuk, Haorui Li, Rishi Thakkar, Ben Hamm, Yuanzhe Li, Xue Huang, Wenpeng Wu, Anish Shanbhag, Harry Kim, Chuan Chen, and Junjie Lai. AIConfigurator: Lightning-fast configuration optimization for multi-framework LLM serving, 2026.

Dongha Yoon, Younghoon Min, Hoshik Kim, Sam H. Noh, and Jongryool Kim. TraCT: Disaggregated llm serving with cxl shared memory kv cache at rack-scale, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of Large Language Models, 2023.

Zihan Zhao, Baotong Lu, Shengjie Lin, Yizou Chen, Jing Liu, Yanqi Zhang, Ziming Miao, Ming-Chang Yang, Haiying Shen, Qi Chen, and Fan Yang. Unifying sparse attention with hierarchical memory for scalable long-context llm serving, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Xianzhe Zheng, Zhengheng Wang, Ruiyan Ma, Rui Wang, Xiyu Wang, Rui Chen, Peng Zhang, Sicheng Pan, Zhangheng Huang, Chenxin Wu, Yi Zhang, Bo Cai, Kan Liu, Teng Ma, Yin Du, Dong Deng, Sai Wu, Guoyun Zhu, Wei Zhang, and Feifei Li. Adaptive multi-objective tiered storage configuration for kv cache in llm service, 2026.

Table S1: Notation (coefficients measured on the platform; estimation protocol in Appendix 10.2). Moved here from the main paper for page budget; the main text points to it here.

Symbol	Meaning
$G = (V, E)$, B_e, M_v, F_v	device graph; per-edge bandwidth; per-device memory/compute
ctx, $T_{\text{split}}/T_{\text{replica}}$	context length (true tokens); per-request latency, Eq. (2): $A_m + b_m$ ctx $+ c_m$ ctx^2
$\tau = b_S - b_R$, $\Delta A = A_R - A_S$	per-token transport tax (measured split relay gap); replica's fixed "setup premium"
$\Delta c = c_S - c_R$, ctx$^* \approx \Delta A/\tau$	transport-growth curvature gap; mode-crossover root ($D' = \tau + 2\Delta c$ ctx > 0)
ϵ_t; k, r, B_{eff}	per-step cost-model residual ($2\epsilon_t$ = noise band); device count; heterogeneity ratio; effective bandwidth

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized LLM serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 193–210. USENIX, 2024.

Qihui Zhou, Peiqi Yin, Pengfei Zuo, and James Cheng. SparseServe: Unlocking parallelism for dynamic sparse attention in long-context llm serving, 2025. ACM International Conference on Supercomputing (ICS) 2026; DOI 10.1145/3797905.3800528.

This supplement expands material that was compressed or inlined in the main paper for page budget. Cross-references of the form "main paper, Section X" point to the submitted main text.

- **Appendix 1** — full proofs of the UCB and Thompson Sampling regret bounds stated without proof in the main paper (Section 5, result (v)).
- **Appendix 2** — step-by-step derivation of the EWMA convergence theorem (main paper, Section 5, result (iv)), including the stationary variance $\sigma^2(1-\alpha)/(1+\alpha)$.
- **Appendix 3** — per-cell significance tests behind the pooled statistics quoted in the main paper (Sections 6.1, 6.2, 6.5, and 6.8).
- **Appendix 4** — the measured cost-model regime map as a complete grid: all 24 grid cells, both held-out points, and the pipeline configuration (23 measured points in total; main paper, Section 6.4).
- **Appendix 6** — ExLlamaV3 baseline protocol details: the needle-arm erratum (both needle arms in fact executed stock RoPE) and the Pride and Prejudice perplexity protocol (this supplement).
- **Appendix 9** — the ablation table that was inlined as prose in the main paper (Section 6.2, formerly `tab:ablation`).
- **Appendix 10** — stack-roofline derivations, the five-tier state taxonomy, the greedy-selection pseudocode, and the bandit-selector equations compressed out of the main paper's Method section; *it also gives the formal derivation from the stack-roofline max to the additive polynomial and the $\tau/\Delta A$ identification protocol (Appendix 10.2).*
- **Appendix 13.2** — full-corpus feasibility-gate false-positive/false-negative audit (375 cells; per-concurrency activation term identified as the gap).
- **Appendix 15.3** — load-aware crossover sensitivity surface over bandwidth $\times$ batch pressure $\times$ arrival rate.
- **Appendix 16.1** — online bootstrap recalibration of τ (convergence, variance, drift-alarm cadence).
- **Appendix 20.1** — closed forms for k-way splits and heterogeneous links.
- **Appendix 25** — online bandit probe on the live llama.cpp stack (real-world confirmation that exploration does not pay).
- **Appendix 26** — fresh same-session n=6 crossover-neighborhood grid.
- **Appendix 27.1** — native-Linux planner-vs-static isolation (fully matched stack; the decision layer's contribution).
- **Appendix 28** — gate safety-margin ROC and batch-penalty portability across engines/OSes; k=3 hybrid-mode closed forms (Appendix 28.1).

- **Appendix 11** — proof of the cumulative prediction-error regret bound (main paper, Section 5, result (ii)).
- **Appendix 12** — the scaling-law derivation, proof, and scope analysis, the parametric sensitivity table across seven topologies, and the supporting transport telemetry (main paper, Section 5, result (iii), and Section 6.5).
- **Appendix 13** — the full cross-model comparison table and per-cell numbers (main paper, Section 6.7).
- **Appendix 17** — the full ten-selector bandit table and the per-round regret curves (main paper, Section 6.6).
- **Appendix 18** — the feasibility, prefix-reuse, and needle hit-rate tables quoted numerically in the main paper (Sections 6.1–6.4 and 6.8).
- **Appendix 30** — the full experimental protocol (true-token prompt construction, actual token counts, nonce, the process-cold and first-at-length one-time taxes) and the needle-probe protocol (depth grid, cold/warm repetition structure, per-cell records).
- **Appendix 32** — the attention-entropy and baseline-latency figures referenced numerically in the main paper.
- **Appendix 33** — the short-context bimodality regression check in full.

Unless stated otherwise, all measurement numbers below are drawn from the released measurement bundle accompanying this submission (per-repetition JSON records under `reports/` and runner scripts under `scripts/`). Every pooled statistic quoted in the main paper was recomputed from those records while drafting this supplement and matches the main paper; the per-cell breakdowns use the same records. Warm repetitions exclude the first repetition of each cell, which carries the process-cold or first-at-length reconfiguration tax defined in the main paper (Section 6.1).

1 Full proofs: sublinear regret of UCB and Thompson Sampling

This appendix expands the standard proofs behind the UCB and Thompson Sampling regret bounds of the main paper (Section 5, result (v)), which were omitted there for space. Nothing here is new: the UCB argument is the finite-time analysis of Auer et al. (2002), and the Thompson Sampling argument is the Bayesian regret decomposition of Russo et al. (2018); both are cited in the main paper. We give the proofs in the paper's mode-selection notation so that the instantiation is fully checkable.

1.1 Setup and concentration lemma

There are K modes (arms). In round t the selector picks arm m_t and observes a stochastic score with mean μ_m; by the sub-Gaussian noise assumption of the main paper (Section 5) the score of every arm is σ^2-sub-Gaussian. Let $m^* = \arg\max_m \mu_m$, $\Delta_m = \mu_{m^*} - \mu_m$, and let $N_m(T)$ count pulls of arm m up to round T. The (expected) cumulative regret is

$$R(T) \;=\; \sum_{m \neq m^*} \Delta_m \, \mathbb{E}[N_m(T)]. \tag{S1}$$

Lemma 1.1 (Sub-Gaussian concentration)**.** *Let $X_1, \dots, X_n$ be i.i.d. σ^2-sub-Gaussian with mean μ and empirical mean $\bar{X}_n$. Then for every $\varepsilon > 0$,*

$$\Pr\big(\bar{X}_n - \mu \geq \varepsilon\big) \;\leq\; \exp\!\Big(-\tfrac{n\varepsilon^2}{2\sigma^2}\Big), \qquad \Pr\big(\bar{X}_n - \mu \leq -\varepsilon\big) \;\leq\; \exp\!\Big(-\tfrac{n\varepsilon^2}{2\sigma^2}\Big). \tag{S2}$$

Proof. By the Chernoff method and the sub-Gaussian moment bound $\mathbb{E}e^{\lambda(X_i-\mu)} \leq e^{\lambda^2\sigma^2/2}$: independence gives $\mathbb{E}e^{\lambda n(\bar{X}_n-\mu)} \leq e^{n\lambda^2\sigma^2/2}$, so Markov's inequality yields $\Pr(\bar{X}_n - \mu \geq \varepsilon) \leq e^{-n\lambda\varepsilon + n\lambda^2\sigma^2/2}$; optimizing at $\lambda = \varepsilon/\sigma^2$ gives the first bound, and the second follows by applying the first to $-X_i$. □

1.2 UCB: THE FINITE-TIME ANALYSIS

The UCB selector of the main paper (Section 4, Method) chooses $m_t = \arg\max_m \{\bar{S}_m(t) + c_{t,N_m(t)}\}$ with confidence radius $c_{t,n} = \beta\sqrt{\log t/n}$ and $\beta = \sqrt{2\sigma^2}$. We prove the regret bound for the one-parameter family $c_{t,n} = \sqrt{2\alpha\sigma^2 \log t/n}$, $\alpha \geq 4$; the paper's radius is the $\alpha \to 1$ member of this family, and Remark 1.3 explains how the standard refinement tightens the leading constant to the form quoted in the main paper.

Theorem 1.2 (UCB finite-time bound (Auer et al., 2002) in σ^2 form)**.** *Under σ^2-sub-Gaussian scores, the UCB selector with radius $c_{t,n} = \sqrt{2\alpha\sigma^2 \log t/n}$, $\alpha \geq 4$, satisfies, for every suboptimal arm m,*

$$\mathbb{E}[N_m(T)] \;\leq\; \frac{8\alpha\sigma^2 \log T}{\Delta_m^2} \;+\; 1 \;+\; \frac{\pi^2}{3}, \tag{S3}$$

and therefore

$$R(T) \;\leq\; \sum_{m \neq m^*} \Big(\frac{8\alpha\sigma^2 \log T}{\Delta_m} \;+\; \Big(1 + \frac{\pi^2}{3}\Big)\Delta_m \Big). \tag{S4}$$

For rewards in $[0,1]$ ($\sigma^2 = \frac{1}{4}$) and $\alpha = 4$ this is exactly the bound $8\log T/\Delta_m + (1+\pi^2/3)\Delta_m$ of Auer et al. (2002, Theorem 1).

Proof. Fix a suboptimal arm m. Write $\bar{S}_{m,n}$ for the empirical mean of arm m over its first n pulls. If arm m is chosen in round t with $N_m(t-1) = n$ and $N_{m^*}(t-1) = n^*$, then $\bar{S}_{m,n} + c_{t,n} \geq \bar{S}_{m^*,n^*} + c_{t,n^*}$, so at least one of the following three events holds:

(i) $\bar{S}_{m^*,n^*} \leq \mu_{m^*} - c_{t,n^*}$ (the optimal arm is under-estimated);

(ii) $\bar{S}_{m,n} \geq \mu_m + c_{t,n}$ (arm m is over-estimated);

(iii) $\Delta_m < 2\, c_{t,n}$ (the gap is below twice the radius).

Indeed, if (i) and (ii) both fail, then $\mu_{m^*} \leq \bar{S}_{m^*,n^*} + c_{t,n^*} \leq \bar{S}_{m,n} + c_{t,n} \leq \mu_m + 2c_{t,n}$, so $\Delta_m \leq 2c_{t,n}$, which is (iii).

Bounding (iii). $\Delta_m \geq 2c_{t,n} = 2\sqrt{2\alpha\sigma^2 \log t/n}$ as soon as $n \geq 8\alpha\sigma^2 \log t/\Delta_m^2$. Hence, with $\ell = \lceil 8\alpha\sigma^2 \log T/\Delta_m^2 \rceil$, event (iii) is impossible for $n \geq \ell$ at every $t \leq T$.

Bounding (i) and (ii). By Lemma 1.1 with $\varepsilon = c_{t,n}$, each of (i) and (ii) has, for fixed (t,n), probability at most $\exp\big(-nc_{t,n}^2/(2\sigma^2)\big) = t^{-\alpha}$. Summing over all pairs (n^*, n) with $1 \leq n^* \leq t$ and $\ell \leq n \leq t$,

$$\mathbb{E}[N_m(T)] \;\leq\; \ell + \sum_{t=1}^{T} \sum_{n^*=1}^{t} \sum_{n=\ell}^{t} 2\, t^{-\alpha} \;\leq\; \ell + 2\sum_{t=1}^{T} t^{2-\alpha}. \tag{S5}$$

For $\alpha = 4$ the remaining sum is $2\sum_{t\geq 1} t^{-2} \leq \pi^2/3$, and absorbing the ceiling into the $+1$ gives the stated per-arm bound. Summing $\Delta_m\, \mathbb{E}[N_m(T)]$ over $m \neq m^*$ via Eq. (S1) completes the proof. □

Remark 1.3 (Tightening to the constant quoted in the main paper). Theorem 1.2 uses the conservative radius with $\alpha \geq 4$ so that the union bound over the t^2 index pairs (n^*, n) converges. The refined analysis—a peeling (doubling) argument over pull counts together with the anytime confidence schedule $c_{t,n} = \sqrt{2\sigma^2 \log f(t)/n}$, $f(t) = 1 + t\log^2 t$ (Auer & Ortner, 2010) and (Lattimore & Szepesvári, 2020, Chapter 7)—tightens the leading term to $8\sigma^2 \log T/\Delta_m + O(1)$ per arm, i.e.

$$R(T) \;\leq\; \sum_{m \neq m^*} \frac{8\sigma^2 \log T}{\Delta_m} \;+\; O(1), \tag{S6}$$

which is the form quoted in the main paper (Section 5, result (v)). The implementation uses the $\alpha \to 1$ radius $\beta\sqrt{\log t/N_m(t)}$ with $\beta = \sqrt{2\sigma^2}$; all of these radii differ only in lower-order logarithmic factors, and the per-arm regret shape $\Theta(\sigma^2 \log T/\Delta_m)$ is common to every variant.

Corollary 1.4 (Worst-case (distribution-free) bound)**.** *For K arms, $R(T) \le 2\sqrt{8\sigma^2 KT\log T} + O(K) = O\big(\sqrt{KT\log T}\big)$.*

Proof. Split the arms at a threshold $\varepsilon > 0$. Arms with $\Delta_m \le \varepsilon$ contribute at most $T\varepsilon$ in total. Each arm with $\Delta_m > \varepsilon$ contributes at most $8\sigma^2 \log T/\Delta_m + O(\Delta_m) \le 8\sigma^2 \log T/\varepsilon + O(\Delta_m)$, so these at most K arms contribute at most $8\sigma^2 K \log T/\varepsilon + O(K)$. Balancing the two terms at $\varepsilon = \sqrt{8\sigma^2 K \log T/T}$ gives $R(T) \le 2\sqrt{8\sigma^2 KT\log T} + O(K)$. □

1.3 Thompson Sampling: the Bayesian regret decomposition

The implementation (main paper, Section 4, Method) uses a Gaussian posterior per arm: after n pulls of arm m the posterior is $\mathcal{N}\big(\bar{S}_{m,n},\, \sigma^2/n\big)$ — mean equal to the empirical average, variance proportional to the inverse of the observation count. This is the exact posterior under a Gaussian likelihood with known σ^2 and an improper flat prior. In each round the selector samples θ_m from every arm's posterior and plays $m_t = \arg\max_m \theta_m$. We prove the regret bound under the Bayesian view in which the true means $\mu = (\mu_1, \dots, \mu_K)$ are themselves drawn from the prior; the resulting *Bayes regret* $\mathrm{BR}(T) = \mathbb{E}\big[\sum_{t=1}^{T}(\mu_{m^*} - \mu_{m_t})\big]$ matches the frequentist regret under any well-specified instance and yields the worst-case bound stated in the main paper.

Lemma 1.5 (Posterior-sampling identity, after Russo et al. (2018))**.** *Conditioned on the history H_{t-1}, the played arm m_t and the true optimal arm m^* are identically distributed. Consequently, for any H_{t-1}-measurable upper-confidence sequence $U_t(\cdot)$,*

$$\mathbb{E}\big[U_t(m_t) \,\big|\, H_{t-1}\big] \;=\; \mathbb{E}\big[U_t(m^*) \,\big|\, H_{t-1}\big]. \tag{S7}$$

Proof. Under the Bayesian view, $\mu \mid H_{t-1}$ is distributed as the posterior. Thompson Sampling draws $\theta \sim$ posterior independently and plays the argmax of θ; hence $\Pr(m_t = m \mid H_{t-1}) = \Pr(m = \arg\max_j \theta_j \mid H_{t-1}) = \Pr(m = m^* \mid H_{t-1})$, because θ and μ have the same conditional law. Any H_{t-1}-measurable function of the argmax therefore has the same conditional expectation for m_t as for m^*. □

Theorem 1.6 (Gaussian Thompson Sampling; width bound)**.** *With upper and lower confidence sequences $U_t(m) = \bar{S}_{m,N_m(t-1)} + r_{t,N_m(t-1)}$ and $L_t(m) = \bar{S}_{m,N_m(t-1)} - r_{t,N_m(t-1)}$, where $r_{t,n} = \sqrt{2\sigma^2 \log(T^4)/n} = \sqrt{8\sigma^2 \log T/n}$,*

$$\mathrm{BR}(T) \le \sum_m \mathbb{E}\Big[\sum_{t:\, m_t = m} 2\, r_{t,N_m(t-1)}\Big] + O(K) \le 4\sqrt{8\sigma^2 KT\log T} + O(K) \;=\; O\big(\sqrt{KT\log T}\big), \tag{S8}$$

and the distribution-dependent form (cited; see the remark below) is

$$\mathrm{BR}(T) \le \sum_{m \ne m^*} O\big(\sigma^2 \log T/\Delta_m\big).$$

Proof. Call the event $\mathcal{G} = \{\, L_t(m) \le \mu_m \le U_t(m)\ \forall\, t \le T,\ \forall\, m \,\}$ the *good event*. By Lemma 1.1 with $\varepsilon = r_{t,n}$, a one-sided failure at fixed (t, m, n) has probability at most T^{-4}; a union bound over $t \le T$, $m \le K$, and $n \le t$ gives $\Pr(\neg\mathcal{G}) \le 2K/T^2$, whose regret contribution is $O(K)$ after bounding regret per round by $\sum_m \Delta_m$.

Lemma 1.5 is applied *unconditionally*: the identity uses only that θ and μ share the posterior law given H_{t-1}, and the good event does not enter it, so $\mathbb{E}[U_t(m_t)] = \mathbb{E}[U_t(m^*)]$. Adding and subtracting U_t and splitting over $\mathcal{G}$ and its complement,

$$\mathbb{E}\big[\mu_{m^*} - \mu_{m_t}\big] = \mathbb{E}\big[\mu_{m^*} - U_t(m^*)\big] + \mathbb{E}\big[U_t(m_t) - \mu_{m_t}\big] \underbrace{\mathbb{E}\big[U_t(m^*) - U_t(m_t)\big]}_{=\,0} \tag{S9}$$

$$\le \mathbb{E}\big[U_t(m_t) - L_t(m_t)\big]\, \Pr(\neg\mathcal{G}) \sum_m \Delta_m = \mathbb{E}\big[2\, r_{t,N_{m_t}(t-1)}\big] + O(K/T), \tag{S10}$$

where on $\mathcal{G}$ both $\mu_{m^*} \ge L_t(m^*)$ and $\mu_{m_t} \le U_t(m_t)$ hold, the width $U_t(m_t) - L_t(m_t) = 2r_{t,N_{m_t}(t-1)}$ is H_{t-1}-measurable, and off $\mathcal{G}$ we use the trivial bound $\sum_m \Delta_m$ with $\Pr(\neg\mathcal{G}) \le$

$2K/T^2$. Summing over rounds gives the first bound. For the second, group rounds by the played arm: the k-th pull of arm m contributes width $2\,r_{t,k} = 2\sqrt{8\sigma^2 \log T/k}$, and $\sum_{k=1}^{N_m} k^{-1/2} \le 2\sqrt{N_m}$, so

$$\mathrm{BR}(T) \;\le\; 4\sqrt{8\sigma^2 \log T}\,\sum_m \mathbb{E}\big[\sqrt{N_m(T)}\big] + O(K) \;\le\; 4\sqrt{8\sigma^2 \log T}\,\sqrt{K\,T} + O(K), \quad \text{(S11)}$$

where the last step is Cauchy–Schwarz, $\sum_m \sqrt{N_m} \le \sqrt{K \sum_m N_m} = \sqrt{KT}$. For the distribution-dependent form we cite rather than re-derive the posterior anti-concentration step: the sharper Gaussian Thompson Sampling analyses (Agrawal & Goyal, 2012; Russo et al., 2018) give $N_m(T) \le 32\sigma^2 \log T/\Delta_m^2$ on $\mathcal{G}$, hence per-arm regret $O(\sigma^2 \log T/\Delta_m)$. The self-contained argument above establishes the worst-case $O(\sqrt{KT \log T})$ rate; the improvement to a $\log T$ per-arm rate is exactly what the cited anti-concentration argument supplies, and we do not claim to re-prove it here. □

Remark 1.7 (Constants and the quoted form). The width argument above is deliberately self-contained; sharper analyses of Gaussian Thompson Sampling (Agrawal & Goyal, 2012; Russo et al., 2018) tighten the per-arm constant to the $8\sigma^2 \log T/\Delta_m$ form quoted in the main paper (Section 5, result (v)), and Gaussian Thompson Sampling is asymptotically optimal with leading term $2\sigma^2 \log T/\Delta_m$. The operative content of the main paper's result (v) is the *shape* of the bound—logarithmic distribution-dependent regret and an $O(\sqrt{KT \log T})$ worst case—which both selectors share, in contrast to the greedy selector's linear regret (main paper, Section 5, result (ii)).

Remark 1.8 (Why the asymptotic bounds do not predict the 500-round measurements). The bounds above are asymptotic: they suppress the constant-factor exploration cost that dominates at moderate horizons, and they assume stationarity. The 20-seed simulation study in the main paper (Section 6.8) quantifies both caveats on score distributions calibrated on the measured configurations: under stationary telemetry the bandit selectors accumulate roughly twice the greedy selector's regret over 500 rounds (the exploration cost of Theorem 1.2's forced sampling of every feasible arm), and under abrupt regime shifts every running-average estimator—including UCB and Thompson Sampling—lags the shifted scores, so the practical lever is recency-weighting rather than exploration. Appendix 3 (Table S6) gives the per-selector significance tests.

2 Full derivation: EWMA convergence of the online cost model

This appendix expands the proof of the EWMA convergence theorem (main paper, Section 5, result (iv)) step by step. Recall the setup. The cost model's transport estimate is updated online by an exponential weighted moving average (EWMA): with $\hat{V}_t$ the predicted visible transport at time t, V_t^{obs} the observed value, prediction error $e_t = V_t^{\mathrm{obs}} - \hat{V}_t$, and decay factor $\alpha \in [0, 1)$ ($\alpha = 0.9$ in the implementation),

$$b_{t+1} = \alpha b_t + (1-\alpha)\, e_t, \qquad \hat{V}_{t+1} = \hat{V}_t + b_{t+1}, \quad \text{(S12)}$$

where b_t is the EWMA bias correction. Within a stationary segment of length n starting at t_0, suppose $\mathbb{E}[e_t] = \mu + \delta_t$ with $|\delta_t| \le L\,\Delta_w$ (the Lipschitz assumption of the main paper, Section 5), and, under the model for which the bound is stated, the e_t are treated as uncorrelated with $\mathrm{Var}(e_t) \le \sigma^2$. This is a modelling assumption, not a claim about the raw traces: a predictor that is an EWMA of past observations can itself induce negative autocorrelation in e_t, so the variance below is a model quantity; the main paper's calibration paragraph reports the measured stationary scatter (0.229σ) against which we check it.

Lemma 2.1 (Unrolling). *For every $n \ge 1$,*

$$b_{t_0+n} \;=\; \alpha^n\, b_{t_0} \;+\; (1-\alpha)\sum_{i=0}^{n-1} \alpha^i\, e_{t_0+n-1-i}. \quad \text{(S13)}$$

Proof. By induction on n. For $n = 1$, Eq. (S12) is the claim. Assuming the claim for n, apply Eq. (S12) once more:

$$b_{t_0+n+1} = \alpha\, b_{t_0+n} + (1-\alpha)\, e_{t_0+n} \tag{S14}$$

$$= \alpha\Big(\alpha^n b_{t_0} + (1-\alpha)\sum_{i=0}^{n-1}\alpha^i e_{t_0+n-1-i}\Big) + (1-\alpha)\, e_{t_0+n} \tag{S15}$$

$$= \alpha^{n+1} b_{t_0} + (1-\alpha)\sum_{i=0}^{n-1}\alpha^{i+1} e_{t_0+n-1-i} + (1-\alpha)\, e_{t_0+n} \tag{S16}$$

$$= \alpha^{n+1} b_{t_0} + (1-\alpha)\sum_{j=0}^{n}\alpha^{j} e_{t_0+n-j}, \tag{S17}$$

where the last line re-indexes $j = i + 1$ in the sum and folds in the final term; this is the claim for $n + 1$. □

Lemma 2.2 (Weight sums)**.** *For $\alpha \in [0, 1)$ and $n \geq 1$,*

$$(1-\alpha)\sum_{i=0}^{n-1}\alpha^i = 1-\alpha^n, \qquad (1-\alpha)^2\sum_{i=0}^{n-1}\alpha^{2i} = \frac{1-\alpha}{1+\alpha}\left(1-\alpha^{2n}\right). \tag{S18}$$

Proof. The geometric series $\sum_{i=0}^{n-1}\alpha^i = (1-\alpha^n)/(1-\alpha)$ gives the first identity on multiplication by $(1-\alpha)$. The same series with ratio α^2 gives $\sum_{i=0}^{n-1}\alpha^{2i} = (1-\alpha^{2n})/(1-\alpha^2)$; multiplying by $(1-\alpha)^2$ and factoring $1-\alpha^2 = (1-\alpha)(1+\alpha)$ yields $(1-\alpha)^2 \cdot \frac{1-\alpha^{2n}}{(1-\alpha)(1+\alpha)} = \frac{1-\alpha}{1+\alpha}(1-\alpha^{2n})$. □

Theorem 2.3 (EWMA convergence; main paper, Section 5, result (iv))**.** *Under the assumptions above,*

$$\left|\mathbb{E}[b_{t_0+n}]-\mu\right| \;\leq\; L\,\Delta_w \,+\, \alpha^n\left|b_{t_0}-\mu\right|, \quad \mathrm{Var}(b_{t_0+n}) \;=\; \frac{1-\alpha}{1+\alpha}\left(1-\alpha^{2n}\right)\sigma^2 \;\xrightarrow[n\to\infty]{}\; \frac{1-\alpha}{1+\alpha}\,\sigma^2. \tag{S19}$$

Proof. Bias. Take expectations in Lemma 2.1 and substitute $\mathbb{E}[e_t] = \mu + \delta_t$:

$$\mathbb{E}[b_{t_0+n}] = \alpha^n b_{t_0} + (1-\alpha)\sum_{i=0}^{n-1}\alpha^i\,(\mu + \delta_{t_0+n-1-i}) \tag{S20}$$

$$= \alpha^n b_{t_0} + (1-\alpha^n)\,\mu + (1-\alpha)\sum_{i=0}^{n-1}\alpha^i\,\delta_{t_0+n-1-i}, \tag{S21}$$

where the second equality uses the first identity of Lemma 2.2. Subtracting μ and taking absolute values,

$$\left|\mathbb{E}[b_{t_0+n}] - \mu\right| \leq \alpha^n\left|b_{t_0}-\mu\right| + (1-\alpha)\sum_{i=0}^{n-1}\alpha^i\left|\delta_{t_0+n-1-i}\right| \tag{S22}$$

$$\leq \alpha^n\left|b_{t_0}-\mu\right| + (1-\alpha^n)\,L\,\Delta_w \;\leq\; \alpha^n\left|b_{t_0}-\mu\right| + L\,\Delta_w, \tag{S23}$$

using $|\delta_t| \leq L\Delta_w$ and Lemma 2.2 again.

Variance. The deterministic term $\alpha^n b_{t_0}$ does not contribute to the variance. Under the uncorrelated-error model of the setup (and the caveat stated there), the variance of the sum is the sum of the variances:

$$\mathrm{Var}(b_{t_0+n}) = \mathrm{Var}\Big((1-\alpha)\sum_{i=0}^{n-1}\alpha^i e_{t_0+n-1-i}\Big) = (1-\alpha)^2\sum_{i=0}^{n-1}\alpha^{2i}\,\mathrm{Var}(e_{t_0+n-1-i}) \tag{S24}$$

$$\leq (1-\alpha)^2\sigma^2\sum_{i=0}^{n-1}\alpha^{2i} = \frac{1-\alpha}{1+\alpha}\left(1-\alpha^{2n}\right)\sigma^2, \tag{S25}$$

by the second identity of Lemma 2.2; equality holds when $\mathrm{Var}(e_t) = \sigma^2$ throughout the segment. Since $1 - \alpha^{2n} \to 1$ as $n \to \infty$, the stationary variance is $\frac{1-\alpha}{1+\alpha}\sigma^2$, and the formula is exact (not only an upper bound) for i.i.d. noise, so the stationary bound is tight. □

Corollary 2.4 (Stationary scatter and the $\alpha = 0.9$ instantiation)**.** *The stationary standard deviation of the bias is $\sigma\sqrt{(1-\alpha)/(1+\alpha)}$. For $\alpha = 0.9$:*

$$\mathrm{Var}(b_\infty) = \tfrac{0.1}{1.9}\,\sigma^2 \approx 0.0526\,\sigma^2, \qquad \mathrm{sd}(b_\infty) = \sqrt{\tfrac{1}{19}}\,\sigma \approx 0.229\,\sigma. \tag{S26}$$

Equivalently, the EWMA's stationary scatter equals that of a plain average over $(1+\alpha)/(1-\alpha) = 19$ i.i.d. observations. The burn-in after a regime boundary decays as α^n: the initialization error is halved after $n \geq \log(0.5)/\log(0.9) \approx 6.6$ rounds (7 rounds), and reduced below 5% after $n \geq \log(0.05)/\log(0.9) \approx 28.4$ rounds (29 rounds).

Remark 2.5 (Bias–variance tradeoff and the measured staleness). Theorem 2.3 makes the tradeoff explicit: as $\alpha \to 1$ the stationary variance vanishes, but the α^n burn-in after a regime boundary slows, so the estimator tracks change more sluggishly. This is exactly the staleness mechanism measured in the main paper (Section 6.8), where running-average estimators lag post-shift scores and recency-weighted variants repair the lag. Convergence of the EWMA bias also does *not* by itself imply good mode selection: the main paper (Section 6.5) measures a systematic, regime-wide steady-state error in the current instantiation, whose repair is the cost-model refit identified there, not the accumulation of more telemetry.

From the scalar error to the model coefficients. The dispatch loop updates the *transport* estimate online rather than the six fitted coefficients directly. With the fixed-cost gap and the curvature held at their ladder values, each observed request yields a per-token error $e_t = (D_t^{\mathrm{obs}} + \Delta A - \Delta c\,\mathrm{ctx}_t^2)/\mathrm{ctx}_t$, whose EWMA $\hat{\tau}_t$ replaces τ in T_{split}; equivalently $b_S \leftarrow b_R + \hat{\tau}_t$, with A_S, A_R, c_S, c_R unchanged. Only an alarm (a 2σ deviation of the EWMA) or a topology change triggers a full ladder refit, which re-estimates all six coefficients by least squares. This is the precise sense in which "the cost model is updated online": the tunable coordinate is the per-token tax, and the rest is re-fit only when the estimator's own error signal fires.

3 PER-CELL SIGNIFICANCE TESTS

The main paper quotes four pooled significance tests and one simulation test family. This appendix lists every per-cell p value behind them, with the per-repetition records they were computed from. All statistics below were recomputed from the released per-repetition JSON records while drafting this supplement; the pooled values reproduce the main paper exactly.

Methods. Two-arm cells use a two-sided Welch t-test (unequal sample sizes, no equal-variance assumption) on warm repetitions; the first repetition of every cell is excluded (process-cold / first-at-length reconfiguration tax, main paper Section 6.1). Pooled tests are one-sample t-tests over cell-level effect sizes, as stated per table. Per-cell sample sizes are small ($n = 2$ per arm in most cells, so Welch df $\approx$ 1–2): per-cell p values should be read as cell-level evidence, with the pooled test carrying the inferential weight — this is the same framing as the main paper. Significance threshold $\alpha = 0.05$, two-sided, throughout.

3.1 BF16 FEASIBILITY: STAGED RUNTIME VS. HF `DEVICE_MAP="AUTO"`

Main paper, Section 6.1 (feasibility matrix, reproduced in this supplement as Table S43): the staged runtime is faster than HF Auto from 1,024 tokens upward, and the advantage is significant at 2,048 tokens (Welch $t = -4.03$, df$= 3.0$, $p = 0.028$). Table S2 lists every cell. Effect is (tworope − HF)/HF; negative favors the framework. Warm per-repetition totals (ms) are shown in parentheses.

The 3,072 cell illustrates the power problem: the largest effect (-26.5%) is non-significant per cell because the two tworope repetitions bracket a wide spread (7,936–13,220 ms), while the 2,048 cell reaches significance on $n = 4$ vs. 2. Means per cell (5,251 / 5,781 / 6,568 / 10,578 vs. 5,305 / 6,215 / 8,597 / 14,396 ms) match Table S43.

Table S2: Per-cell Welch t-tests, bf16 staged vs. HF Auto (warm cold-prefill total ms, n per arm in the reps columns). The 2,048 cell is the test reported in the main paper; the remaining cells are same-sign but underpowered ($n = 2$ per arm). Source records: `reports/feasibility/runtime_current_bf16.json`, `reports/feasibility/hf_device_map_auto_bf16.json`.

Context (true)	tworope reps (ms)	HF Auto reps (ms)	Effect	Welch t	df	p
512 (523)	5,330 / 5,172	5,302 / 5,308	-1.0%	-0.68	1.0	0.62
1,024 (1,035)	5,848 / 5,715	6,211 / 6,219	-7.0%	-6.49	1.0	0.096
2,048 (2,059)	6,701 / 5,189 / 7,608 / 6,773	8,606 / 8,587	-23.6%	$\mathbf{-4.03}$	3.0	**0.028**
3,072 (3,083)	13,220 / 7,936	14,130 / 14,662	-26.5%	-1.44	1.0	0.38

3.2 Prefix reuse: pooled test over the ten cold/warm pairs

Main paper, Section 6.2 (prefix-reuse table, reproduced in this supplement as Table S44): pooled over the ten non-first cold/warm pairs the prefix-reuse speedup is significant (one-sample $t = 4.19$, df$= 9$, $p = 0.0023$; Wilcoxon signed-rank $p = 0.002$). Table S3 lists all ten pairs. Pair 0 at each length is excluded (first-at-length reconfiguration tax; reported separately in the main paper's protocol). Speedup is (cold $-$ warm)/cold per pair; generation lengths match within each pair (temperature $= 0$).

Table S3: The ten prefix-reuse cold/warm pairs behind the pooled test (one-sample t on pair speedups: $t = 4.19$, df$= 9$, $p = 0.0023$; Wilcoxon signed-rank $W^+ = 55/55$, $p = 0.002$). Source records: `reports/prefix/prefix_bf16.json`.

Prefix (true tokens)	Pair	Cold (ms)	Warm (ms)	Δ (ms)	Speedup
128 (139)	1	3,830.5	3,811.8	18.7	0.5%
128 (139)	2	3,540.6	3,438.9	101.8	2.9%
512 (523)	1	4,956.0	4,418.3	537.6	10.8%
512 (523)	2	4,339.0	3,787.3	551.7	12.7%
1,024 (1,035)	1	5,070.1	3,987.8	1,082.3	21.3%
1,024 (1,035)	2	5,503.2	4,405.6	1,097.6	19.9%
2,048 (2,059)	1	6,536.7	4,408.3	2,128.4	32.6%
2,048 (2,059)	2	6,508.6	4,422.2	2,086.4	32.1%
4,096 (4,107)	1	8,871.0	4,533.7	4,337.3	48.9%
4,096 (4,107)	2	8,418.2	4,222.6	4,195.6	49.8%
Pooled over the 10 pairs (mean speedup $23.15\% \pm 17.45$)				$t = 4.19$, df$= 9$, $p = 0.0023$	

As a robustness check, the same pooled test on absolute savings (Δ ms, mean $1{,}613.7 \pm 1{,}571.9$ ms) gives $t = 3.25$, df$= 9$, $p = 0.010$ — the conclusion is insensitive to whether pairs are compared in relative or absolute terms. Restore fired in all 15 pairs (`restore_reported=true`); the per-length mean speedups (1.7% / 11.8% / 20.6% / 32.3% / 49.4%) match Table S44.

3.3 Crossover band: replica advantage, per cell and pooled

Main paper, Section 6.5 (footnote): pooled across the four crossover-band cells with full repetition structure, the single-card replica's advantage is $+23.3\% \pm 11.4$ (paired $t = 4.12$, df$= 3$, $p = 0.026$; per-cell p: .050/.76/.025/.017). Table S4 assigns those p values to their cells. The split arm is the shared balanced dual-GPU execution (`staged_infer` label); the replica arm is `request_parallel` with one worker. Cell margin is (split $-$ rp)/split; positive favors the replica. Per-cell tests are Welch t on warm repetitions (positive t = replica faster).

The non-significant cell is 4,096: its two replica repetitions are bimodal (7,439 vs. 4,920 ms — the known 4-bit bitsandbytes-path bimodality disclosed in this supplement), so the per-cell comparison is inconclusive even though the pooled verdict over the band is significant. The generation-matched 1,536 cell and the 2,048 cell (large but asymmetric rep structure) are excluded from the pooled test, as in the main paper; see Appendix 4 for the full band.

Table S4: Per-cell Welch tests in the crossover band (4-bit, concurrency 1), and the pooled one-sample t over the four cell margins reported in the main paper. Source records: `reports/costmodel/cells/` (`staged_infer_ctx*_c1.json`, `request_parallel_ctx*_c1.json`).

Context	split reps (ms)	rp reps (ms)	Margin	Welch t	df	p
3,072	5,656 / 5,272	4,273 / 4,072	$+23.6\%$	5.96	1.5	0.050
4,096	6,872 / 6,493	7,439 / 4,920	$+7.5\%$	0.39	1.0	0.76
6,144	8,827 / 8,457	6,262 / 6,108	$+28.4\%$	12.27	1.3	0.025
12,288	16,332 / 15,903	10,735 / 10,619	$+33.8\%$	24.51	1.1	0.017
Pooled over the 4 cell margins ($+23.3\% \pm 11.4$)				$t = 4.12$, df$= 3$, $p = 0.026$		

3.4 Concurrency region: replica vs. staged, per cell and pooled

Main paper, Section 6.4 (mechanism 2): at concurrency ≥ 2 measured replica batches are 38–53% slower than the staged path — 62–113% slower per request (pooled $+83.8\% \pm 24.0$ per request, paired $t = 6.97$, df$= 3$, $p = 0.006$). Table S5 lists every cell. Cell slowdown is (rp $-$ staged)/staged per request; per-cell tests are Welch t on warm per-request times (positive t = staged faster).

Table S5: Per-cell Welch tests in the concurrency region (4-bit, per-request ms), and the pooled one-sample t over the four cell slowdowns reported in the main paper. Source records: `reports/costmodel/cells/` (`request_parallel_ctx*_c{2,4}.json`, `staged_infer_ctx*_c{2,4}.json`).

Context	Conc	rp per-req reps (ms)	staged per-req reps (ms)	Slower	Welch t	df	p
1,024	2	9,778 / 8,391	6,418 / 4,787	$+62.1\%$	3.25	1.9	0.086
1,024	4	5,802 / 5,807	2,737 / 2,707	$+113.2\%$	204.81	1.1	0.0024
2,048	2	10,171 / 10,088	5,282 / 5,195	$+93.4\%$	81.07	2.0	0.0002
2,048	4	6,264 / 6,156	4,083 / 3,385	$+66.3\%$	7.01	1.0	0.083
Pooled over the 4 cell slowdowns ($+83.8\% \pm 24.0$)					$t = 6.97$, df$= 3$, $p = 0.006$		

The sign of the effect is uniform across cells; the two cells with very tight staged repetitions reach per-cell significance on their own, while the pooled test is what the main paper quotes. Batch-level totals behind these per-request figures appear in Appendix 4 (Table S8).

3.5 Bandit selector simulation: per-selector paired tests over 20 seeds

Main paper, Section 6.8: three headline comparisons. All are paired t-tests over the 20 seeds (base seed 42) of the simulation; Table S6 lists them with exact p values recomputed from the released per-seed records (`bandit_regret_detailed.json`).

What the simulated scores are, and are not. The simulation does not consume the cost model's predictions. It replays the *measured* latency distributions of the 24 tested configurations as ground truth (`scripts/bandit_simulation.py`: "the 'true' mean latency for each mode is estimated from the measured latency in the paper", with the measured per-cell means as anchor data), defines the score as $-$latency, and has each selector estimate those means from its own noisy observations. The estimator under study is therefore the selector's running average over observed latencies, not the pre-repair cost model, and the 45.0% mode-selection accuracy that motivated the repair is a property of a different quantity that does not enter this study. The assumptions the simulation does carry are the ones it states: stationarity in the first scenario, sampled abrupt regime shifts plus slow drift in the second, and a declared cold-start perturbation ($\sigma_{\text{init}} = 500$ ms).

These confirm the main paper's three observations verbatim: exploration is a significant cost under stationarity; the running-average selectors are indistinguishable from each other after regime shifts; and sliding-window greedy (W=100) is the only selector that significantly beats greedy under regime shifts ($p < 0.01$) while giving up nothing significant under stationarity ($p = 0.72$).

Table S6: Per-selector paired t-tests over 20 seeds. Positive diff = the listed selector is worse than greedy. Recomputed exact p values are shown; the main paper rounds these tests to $t \approx 11$–12 / $t = -3.76$ / n.s. — the recomputed stationary-cost statistic ($t = 0.36$) and the shift-regime statistic ($t = -3.76$) match the main paper's updated values, with no change to any conclusion.

Regime	Comparison	Diff (ms)	Rel. diff	paired t	df	p
Stationary	UCB (β=1) − Greedy	+21,057	+112.5%	11.30	19	7.1×10^{-10}
Stationary	UCB (β=2) − Greedy	+21,374	+114.2%	11.62	19	4.5×10^{-10}
Stationary	Thompson − Greedy	+21,135	+112.9%	11.36	19	6.5×10^{-10}
Stationary	SW-Greedy (W=100) − Greedy	+1,212	+6.5%	0.36	19	0.72 (n.s.)
Non-stationary	SW-Greedy (W=100) − Greedy	−53,487	−17.1%	−3.76	19	0.0013
Non-stationary	D-UCB (γ=0.95) − Greedy	−37,653	−12.0%	−1.46	19	0.16 (n.s.)
Non-stationary	SW-Thompson (W=100) − Greedy	−30,488	−9.7%	−1.57	19	0.13 (n.s.)
Non-stationary pairwise among the four running-average selectors: $\max \lvert t \rvert = 1.11$ (all n.s.)						

4 Cost-model regime map: the complete grid

Main paper, Section 6.4 reports the cost model's mode-prediction accuracy as a regime roll-up and describes the empirical regime map in prose. This appendix enumerates every measured decision point: the 24-cell grid (20 measured, 4 unmeasured), the 2 held-out points, and the 23rd measured configuration (pipeline, baseline-only). All numbers are from the released evaluation bundle (`reports/costmodel/SUMMARY.md` and `reports/costmodel/cells/`; runner `scripts/run_costmodel_cells.py`; offline scoring script `scripts/eval_cost_model_accuracy.py`). Throughout, "split" denotes the balanced dual-GPU execution shared by the 4-bit `single_gpu` and `staged_infer` plan labels (the semantic trap of mechanism 1 below), and "rp" denotes `request_parallel` with `worker_count=1`, a true single-card full replica.

4.1 Empirical regime map (4-bit, concurrency 1): all 14 measured contexts

Table S7 gives the complete measured map behind the prose description in the main paper (Section 6.5).

Bands. The single-card replica wins **every measured context at which it is feasible, 512 through 24,576 (11/11)**, by margins 4.6–42.4% (significance: Appendix 3.3); it **OOMs from 32,768**, where the split wins by feasibility. There is *no measured latency crossover*: the fitted model's root (769 tokens) lies below the smallest evaluated context, and the fresh n=6 grid's two sub-second split wins at 1,280/1,536 sit inside the $2\epsilon_t$ noise band (Table S53), where the selector follows measurement rather than the fit. The replica feasibility ceiling is bracketed in $(24{,}576, 32{,}768)$.

4.2 Concurrency region (4-bit, concurrency 2/4): staged wins all four cells

Per request, rp is 62–113% slower (pooled $+83.8\% \pm 24.0$, $t = 6.97$, df= 3, $p = 0.006$); on batch totals, rp is 38–53% slower (measured against the rp batch totals, i.e. (rp − staged)/rp). The cost model's replica throughput credit ($\sim \min(\text{conc}, \text{width}) \times 0.9$) has the wrong sign on this topology: replicas pay full prefill per card with no transport hiding, while the staged path amortizes the split's handoffs across the microbatch.

4.3 The full 24-cell grid plus held-out points (23 measured points)

Table S9 enumerates every decision point: the 24 grid cells (20 measured), the 2 held-out points, and the pipeline configuration.

Per-regime roll-up for the pre-repair model (the repaired selector's roll-up is in the main paper, Section 6.4): feasibility 5/6 (8,192 flipped to rp), crossover 2/8 (only the sub-crossover 512/1,024 cells survive), stability 0/4 measured (2 cells unmeasured), long-context 2/2 measured (98,304/131,072 unmeasured). Held-out points were measured *before* any rule fitting: at 24,576 the replica wins by

Table S7: Split vs. single-card replica by measured warm per-request latency, 4-bit, concurrency 1, all 14 measured contexts (`reports/costmodel/accuracy.json`, `regime_map`). Margins are the winner's latency advantage over the loser. The single-card replica wins every context at which it is feasible, 512 through 24,576; the split wins only where the replica is infeasible, so **there is no measured latency crossover on this platform** — the fitted model's root (769 tokens) lies below the smallest evaluated context. An earlier $\leq$1,024 split-wins band came from short-context cells with truncated generations and NF4 bimodality and is superseded by the generation-forced re-measurement. The 1,536 cell is generation-matched (64-token completions; the split's warm repetitions truncated generation early at 36–38 tokens, so the raw ranking compares unequal work). The 4,096 split mean uses the bimodality-free `staged_infer` label (one 21.8 s bimodal outlier under the `single_gpu` label; main paper Section 6.2).

Context	split (ms)	rp (ms)	Winner	Scoring
512	3,064	2,673	**rp**, 12.8% faster	raw
1,024	3,553	3,190	**rp**, 10.2% faster	raw
1,536	4,066	3,445	**rp**, 19.2% faster	gen-matched
2,048	6,122	5,840	**rp**, 4.6% faster	raw
3,072	11,518	7,546	**rp**, 34.5% faster	raw
4,096	12,573	8,632	**rp**, 31.3% faster	raw
6,144	14,698	9,571	**rp**, 34.9% faster	raw
8,192	13,171	7,585	**rp**, 42.4% faster	raw
12,288	21,828	13,976	**rp**, 36.0% faster	raw
16,384	20,741	17,241	**rp**, 16.9% faster	raw
24,576	36,690	21,456	**rp**, 40.4% faster	gen-matched
32,768	64,691	OOM	split (rp infeasible)	raw
49,152	117,198	OOM	split (rp infeasible)	raw
65,536	179,964	OOM	split (rp infeasible)	raw

Table S8: Concurrency cells: measured batch and per-request latency, warm repetitions. The staged path's decode-microbatch interleaving wins all four cells; per-request significance in Appendix 3.4.

Context	Conc	rp batch (ms)	staged batch (ms)	rp per-req (ms)	staged per-req (ms)
1,024	2	18,169	**11,205**	9,084	**5,603**
1,024	4	23,219	**10,888**	5,805	**2,722**
2,048	2	20,259	**10,477**	10,129	**5,238**
2,048	4	24,841	**14,937**	6,210	**3,734**

40.4% as the interval rule predicts; at 1,536 the planner is falsified under both scoring protocols (its `single_gpu` loses either way).

4.4 The single-card replica ceiling

Mechanism 3 of the main paper (Section 6.4) is an unmodeled feasibility boundary: the one-worker replica places the whole 4-bit model plus the full KV cache on one card. The measured single-card peak grows linearly at **191.7 KiB/token** (3.42× the theoretical 56.0 KiB/token bf16 KV slope — activations and allocator overhead scale with context too) plus a 5,382 MiB base, from 5.4 GiB at 512 tokens to 10.0 GiB at 24,576 tokens. The fit projects the OOM boundary exactly where measurement finds it:

The cost model has no per-mode memory-feasibility term for the replica, so it cannot express this boundary through telemetry refit alone; above the replica ceiling the split wins by feasibility (and the planner is correct there for the wrong reason, since its policy never considers rp at concurrency 1 anyway).

Remark 4.1 (Cross-model check (Llama-3-8B, secondary)). On Llama-3-8B the measured 4-bit cells all sit in the split-feasibility region and are $5/5$ correct (main paper, Section 6.7): 8,192 tokens (split 11,357 ms, n=2; HF-auto-4bit infeasible), 16,384 (22,968, n=3), 32,768 (67,185, n=1),

Table S9: Complete scoring grid. Predicted = the current planner's offline prediction (synthetic 2×RTX 3060 relay profile). Hindsight-best = lowest measured warm per-request latency among planner-eligible modes; at 4-bit the `single_gpu` and `staged_infer` labels share one split execution, so either measurement validates both labels. Baselines (HF auto, cpu_offload, llama.cpp, pipeline) are context, never planner modes. Totals: **9/20 = 45.0% correct** on the measured grid (11/20 wrong); **9/22 = 40.9% correct** including held-out points; oracle hindsight 22/22 by construction.

Regime	Context	Quant	Conc	Predicted	Hindsight	OK	Evidence (ms)
feasibility	512	bf16	1	staged	staged	✓	single: infeasible; staged 5,251 (n=2)
feasibility	1,024	bf16	1	staged	staged	✓	single: infeasible; staged 5,782 (n=2)
feasibility	2,048	bf16	1	staged	staged	✓	single: infeasible; staged 6,568 (n=4)
feasibility	3,072	bf16	1	staged	staged	✓	single: infeasible; staged 10,578 (n=2)
feasibility	8,192	4bit	1	single	**rp**	×	split 13,171 (n=2); rp 7,585 (n=2); HF-auto: infeas.
feasibility	65,536	4bit	1	single	split	✓	split 179,964 (n=3); rp: OOM
crossover	512	4bit	1	single	split	✓	split 3,494 (n=2); rp 4,272 (n=2)
crossover	1,024	4bit	1	single	split	✓	split 3,584 (n=2); rp 4,722 (n=2)
crossover	2,048	4bit	1	single	**rp**	×	split 11,005 (n=3); rp 3,406 (n=2)
crossover	3,072	4bit	1	single	**rp**	×	split 5,464 (n=2); rp 4,172 (n=2)
crossover	4,096	4bit	1	single	**rp**	×	split 6,682 (n=2); rp 6,180 (n=2)
crossover	6,144	4bit	1	single	**rp**	×	split 8,642 (n=2); rp 6,185 (n=2)
crossover	12,288	4bit	1	single	**rp**	×	split 16,118 (n=2); rp 10,677 (n=2)
crossover	16,384	4bit	1	single	**rp**	×	split 26,145 (n=2); rp 13,969 (n=2); HF-auto: infeas.
stability	1,024	4bit	2	rp	**split**	×	rp 9,084; staged 5,603 (per-req, n=2)
stability	1,024	4bit	4	rp	**split**	×	rp 5,805; staged 2,722 (per-req, n=2)
stability	2,048	4bit	2	rp	**split**	×	rp 10,129; staged 5,238 (per-req, n=2)
stability	2,048	4bit	4	rp	**split**	×	rp 6,210; staged 3,734 (per-req, n=2)
stability	16,384	4bit	2	rp	—		unmeasured (pipeline baseline only, below)
stability	16,384	4bit	4	rp	—		unmeasured
long-context	32,768	4bit	1	single	split	✓	split 64,691 (n=2); rp: OOM; HF-auto: infeas.
long-context	49,152	4bit	1	single	split	✓	split 117,198 (n=1); rp: OOM; HF-auto: infeas.
long-context	98,304	4bit	1	staged	—		unmeasured
long-context	131,072	4bit	1	staged	—		unmeasured
held-out	1,536	4bit	1	single	**rp**	×	rp 3,041 raw / 3,296 gen-m.; split 4,081 gen-m.; staged raw 2,848 (∼37 gen tok.)
held-out	24,576	4bit	1	single	**rp**	×	rp 21,456; single 35,990; staged 36,690
23rd measured configuration (baseline context, not scored — pipeline is not a planner-selectable mode): pipeline, 16,384 tokens, 4-bit, concurrency 2: $78{,}446 \pm 1{,}300$ ms per 2-request batch (n=3; ≈39 s/req).							

Table S10: Replica feasibility ceiling: forced-run outcomes and the linear peak-memory fit (191.7 KiB/token slope + 5,382 MiB base).

Context	rp status	split (ms)	Projected rp peak (fit)
32,768	OOM (all reps)	64,691	11.25 GiB
49,152	OOM (all reps)	117,198	14.24 GiB
65,536	OOM (all reps)	179,964	17.24 GiB

49,152 (151,366, n=2), 65,536 (226,868, n=3). Llama bf16 cells are partial coverage only (single-device Llama bf16 was never run: ∼14.9 GiB of weights exceed one 12 GiB card, so the staged path is the de facto answer; staged measured 4,614 / 4,837 / 7,184 ms at 512 / 1,024 / 3,072 tokens).

4.5 The three mechanisms, restated compactly

For convenience, the three identified error mechanisms (main paper, Section 6.4): (1) *execution-label mismatch* — the 4-bit `single_gpu` plan label executes a balanced staged-split, not one card, so the cost model's zero-transport single-card scoring is applied to an execution that pays relay transport on every token, and the only true single-card execution (the one-worker replica) is excluded by planner policy at concurrency 1; this inverts every prediction in the 1,536–24,576 band. (2) *wrong-sign concurrency credit* — replicas are credited with a batching benefit they do not have on this topology; measured per-request they are 62–113% slower (Table S5). (3) *unmodeled feasibility ceiling* — Table S10. The refit repairing all three mechanisms—and the fresh cells that falsified the

first repair—are recorded in Section 5. The fitted interval rule (rp iff $1{,}536 \leq$ ctx $\leq 24{,}576$ at 4-bit/concurrency-1) restates the measured map; on the full 24-point decision set it now scores only 20/24, while the mechanistic refit scores 23/24.

5 SELECTOR REFIT: CALIBRATION, REPAIR, AND FRESH VALIDATION

This appendix records the mechanism-level selector repair of the main paper's Section 6.5. All artifacts reproduce from the committed scripts: `scripts/calibrate_selector_latency_model.py`, `scripts/loco_residual_analysis.py`, `scripts/eval_cost_model_accuracy.py`, `scripts/verify_mode_crossover_law.py`, `scripts/run_costmodel_cells.py`, `scripts/run_refit_validation_suite.ps1`, `scripts/run_refit_genforced_suite.ps1`, and `scripts/run_ruler_needle.py`.

Scope of the $2\epsilon_t$ bound, and why the pre-repair failures fall outside it. The guarantee of Section 5, result (ii) bounds the greedy selector's per-step suboptimality by twice the *cost model's* prediction error. Its premise is that the action set is correctly implemented: it compares the selector's choice against the true best mode *among the modes the runtime actually executes*. The three mechanisms diagnosed here violate that premise rather than the error term. An execution label that mapped to a different runtime behaviour, a concurrency term with the wrong sign, and a missing per-mode feasibility ceiling are *model* errors, not transport noise, so $2\epsilon_t$ never bounded them and the guarantee does not describe the pre-repair selector. After the repair the premise holds, the reported prediction error is the fit RMS ($\epsilon_t = 1{,}383$ ms), and the bound applies to the selector as deployed; the re-scored accuracy (21/22 on the grid, 23/24 across all measured decision points, 0 ms observed suboptimality) is its empirical counterpart.

5.1 LATENCY-MODEL CALIBRATION

The score becomes predicted per-request latency $T = A + b\,\text{ctx} + c\,\text{ctx}^2$ per mode (main paper, Eq. (2)). The coefficients are least-squares fits on the measured regime-map/ladder points plus the fresh generation-forced short-context cells (count-style prompts, 63–64-token generations, n=3–4 warm reps), excluding the documented NF4-bimodal 2,048 split cell: $A_S = 2{,}331.5$ ms, $A_R = 2{,}615.3$ ms, $b_S = 0.9290$ ms/token, $b_R = 0.5597$ ms/token, $c_S = 2.752\times10^{-5}$, $c_R = 8.339\times10^{-6}$ ms/token^2; RMS residual 1,382.6 ms, max |relative| 18.3% (`reports/selector_refit/calibration.json`). The measured regime law (replica wins every feasible 4-bit/concurrency-1 context from 512 tokens; 14/14 cells, four fresh) applies inside the estimator's noise band on the calibration platform only.

5.2 LEAVE-ONE-CONTEXT-OUT VALIDATION OF THE LATENCY MODEL

Fifteen refits, each dropping one distinct context from the 26-point calibration (14 split + 12 replica) and re-fitting with the same $b_R \geq 0$ clamp rule; each refit predicts only its held-out point(s) (`scripts/loco_residual_analysis.py`; `reports/selector_refit/loco_crossval.json`). Held-out RMS 2,652.7 ms vs. 1,382.6 ms in-sample; worst |relative| 20.5% (the 4,096 replica cell). The hardest extrapolation—the 65,536 endpoint, fitted through 49,152 only—is reproduced to -5.5% (9.9 s on 179.96 s). Nine of fifteen folds hold within $\pm$10%; the six outliers are 512 replica (-11.1%), 2,048 replica (-13.1%), 8,192 split ($+11.5\%$), 12,288 split (-12.5%), 24,576 split (-18.6%), and 4,096 replica ($+20.5\%$, the worst).

Two readings follow. *Latency* predictions generalize: every fold's held-out error stays within 20.5% relative, including the longest extrapolation, so the per-request latency scores the selector compares are robust to the deletion of any single calibration context. *Constants* trade off: ΔA stays within $+201..+608$ ms of the full-fit $+283.8$ ms when any single short/mid context (512–24,576) is deleted—it is pinned by the generation-forced short cells—but collapses or flips sign when a long-ladder point (32K–65K) is deleted; τ stays within 0.292–0.474 ms/tok when any point *except* 65,536 is deleted (deleting 65K halves it to 0.204). The law's ctx$^* = 769$ is therefore a *joint-calibration* quantity: its per-fold spread is wide (120–1,281, undefined where $\Delta A \leq 0$), and its operational validity rests on the measured regime cells and the held-out grids (21/22 grid / 23/24 across all 24 points, 44/47 off-grid, main paper Section 6.5), not on intercept identifiability. We report this rather

Table S11: Per-fold refits: transport tax τ, replica fixed-cost premium $\Delta A = A_R - A_S$, law crossover $\text{ctx}^*_{\text{lin}} = \Delta A/\tau$ ("—" where $\Delta A \leq 0$), and held-out errors (s split, r replica). Full-fit values: $\tau = 0.369$, $\Delta A = 283.8$ ms, $\text{ctx}^*_{\text{lin}} = 769$.

held-out ctx	τ (ms/tok)	ΔA (ms)	$\text{ctx}^*_{\text{lin}}$	**held-out errors**
512	0.387	+409	1057	s +9.8%, r −11.1%
1,024	0.373	+334	894	s +8.0%, r −0.3%
1,280	0.363	+273	750	s +7.2%, r +7.6%
1,536	0.374	+340	910	s +7.0%, r −1.7%
2,048	0.378	+358	947	r −13.1%
3,072	0.370	+319	860	s +0.4%, r −6.5%
4,096	0.383	+201	525	s +1.4%, r +20.5%
6,144	0.361	+235	652	s −5.5%, r −3.5%
8,192	0.355	+388	1093	s +11.5%, r −3.0%
12,288	0.379	+202	532	s −12.5%, r −1.0%
16,384	0.345	+270	783	s +5.5%, r −0.6%
24,576	0.474	+608	1281	s −18.6%, r +3.3%
32,768	0.292	−3	—	s +5.3%
49,152	0.321	+39	121	s +3.3%
65,536	0.204	−316	—	s −5.5%

than hide it: on this platform the crossover's exact location is the least-identified parameter of the model, and the selector's decisions do not depend on it—the regime law and the feasibility gate do the work in its neighborhood.

5.3 Calibration-error sensitivity of the selector

To bound how a field deployment's imperfect calibration degrades decisions, we perturb the two crossover-law constants—the transport tax τ and the replica fixed-cost premium ΔA—by $\pm 5/10/15\%$ and re-evaluate the split-vs-replica sign at every measured context (`scripts/selector_sensitivity.py`; `reports/selector_refit/sensitivity.json`). The unperturbed quadratic crossover sits at 740 tokens, between the 512 and 1,024 measured cells; under $\pm 15\%$ on either constant it moves only to 632–859 tokens (τ alone: 649–859; ΔA alone: 632–847), so *no measured decision flips*. The first flip requires a $\approx 31\%$ underestimate of τ or a $\approx 32\%$ underestimate of ΔA—more than double the perturbation band. Two structural reasons: (i) the nearest measured cells to the crossover are a factor $\geq 1.4\times$ away, so the decision is a margin away from the boundary, not on it; and (ii) the feasibility-dominated cells (32K–65K and the concurrency region) are decided by the memory gate and the batch-efficiency factor, which are independent of τ and ΔA. Calibration error therefore degrades the *location* of the crossover long before it degrades any *decision*, and the EWMA/regime-law fallback inside the noise band (main paper, Section 6.5) is the operational safeguard for exactly this regime.

5.4 Fresh out-of-sample cells (falsification-repair record)

Cells measured *after* the first repair was frozen; each falsification drove one mechanism fix:

5.5 Re-score (complete)

`scripts/eval_cost_model_accuracy.py`: 24-cell grid 21/22 at n=5 warm reps (was 45.0% correct); all 24 measured decision points 23/24; legacy held-out 2/2; fresh refit held-out 4/4 (26,000/28,000 tokens, 4,096/8,192 $\times$ concurrency 8); Llama 2/2. The single miss is 2,048$\times$concurrency-1, where the true single-card execution wins 3.53 s vs. 5.84 s replica (and 6.12 s staged)—a fourth mechanism: at small contexts single-card dominates both two-GPU modes, which the split-vs-replica refit under-predicts. Counterfactuals on the same 24 points: oracle 24/24 (upper bound, not a policy); the fitted interval band rule 20/24; and *a correctly implemented one-line static rule* (conc$\geq$2 $\rightarrow$ split; conc=1, 4-bit $\rightarrow$ replica iff the single-card replica is feasible at that context, else split; conc=1, bf16 $\rightarrow$ split—single-card not loadable) at **20/22** on the measured grid and

Table S12: Fresh validation cells and the repairs they drove.

Fresh cell	v1 prediction	Measured	Repair
28,000 rp	feasible + winner	OOM (3/3)	ceiling reserve (5%)
16,384 × c2	rp (hysteresis)	staged -19.3%	staged batch-efficiency factor
16,384 × c4	staged feasible	both OOM	concurrency-saturating activation term
Llama rp 8,192/16,384/24,576	2 of 3	rp wins 8K/16K; OOM 24K	per-model KV-slope factor ($1.91\times$)
4,096 × c8	—	staged -4.5% (act 2.5 GiB)	saturating term recalibration
8,192 × c8	—	rp OOM; staged fits	per-worker replica ceiling

23/36 on the deduplicated randomized off-grid set (`scripts/static_rule_baseline.py`; `reports/diagnostics/static_rule_baseline.json`).

The static baseline's two grid misses are exactly the below-crossover short contexts (512/1,024×c1, both split-wins—the converse of the calibrated layer's own single miss); its off-grid misses are *all* feasibility cells (14 both-modes-OOM at ≥14,336×c≥3, which the static rule cannot reject because it has no memory gate, plus one 20 ms tie). The isolation reads: the calibrated layer's measurable increment over the correct static rule is (i) the short-context split cases the c1→replica blanket rule gets wrong, and (ii) the feasibility gate's OOM rejections; the 2,048×c1 single-card case is the one cell where the static rule is right and the calibrated fit misses. This answers the concern that the repairing selector was strawmanned: the baseline is the *correctly stated* one-line rule, and the decision layer still adds 1/22 grid cells (the 512/1,024 short-context splits) and the 10 off-grid cells that are entirely the gate's rejections—its marginal value over the static rule is the calibrated crossover short-side and memory-feasibility rejection.

5.6 Breadth measurements

Decode-length matrix (8,192 tokens, generation-forced, n=4 warm): the replica wins at every decode length—margins -19.4% (128), -16.4% (256), -12.6% (512); marginal decode cost 34.8 (replica) vs. 38.7 (split) ms/token. RULER-style randomized needles (Hsieh et al., 2024) at 16,384 (seed 20260815): 30/30 hits—20 randomized single-needle cells (values 1,327–9,817, depths 7.3–90.0%) plus 10 four-needle distractor cells. Entropy profiling (corrected computation): one 0.78 s eager-attention forward pass; per-layer entropies 2.31–2.69 for layers 0–3 and 1.06–1.95 for layers 4–27 (a regenerated profile under the same corrected computation, 48 tokens, gives 1.83–2.22 / 0.95–1.70 with the same layers-0–3 maximum; `reports/attention_entropy_profile_corrected.json`; the committed script's double-softmax degenerated the earlier profile to $\sim \ln(\text{seq})$; erratum recorded in this supplement).

5.7 External-validity axes without a second machine

Measured effective-bandwidth sweep (load-throttled PCIe). To test the linear $\text{ctx}^*(B)$ law without a second machine, we sweep the background load on the shared PCIe root complex and remeasure the replica-vs-staged cells at each operating point. A rate-limited pinned-copy injector (`scripts/pcie_contention_throttled.py`, an extension of the saturating injector of the contention axis) sustains an aggregate H2D/D2H load $L \in \{0, 6, 12, \text{saturating}\}$ GB/s; the achieved rates are logged per level. Under each level, the forced cells at 8,192 and 16,384 tokens (concurrency 1, 4-bit, n=3, identical protocol) are measured in both modes and written to `reports/costmodel/contention/bw_sweep/`. The effective relay bandwidth is $B_{\text{eff}} = B_0\,(1 - L/B_{\text{sat}})$ with $B_0 = 13.3$ GB/s and B_{sat} the injector's saturated aggregate rate. The transport tax per level is read off the two-context slope with the quadratic term held at its calibration value, and the implied crossover is $\text{ctx}^*(L) = \Delta A/\tau(L)$.

What the sweep shows, and its noise. Three signals are robust. First, the split's 16,384-token latency degrades monotonically with load (21.5 s baseline; 22.4 s at 6 GB/s; 31.9 s at 12 GB/s; 39.2–40.0 s saturated across two passes), while at 8,192 the effect is confined to saturation (13.2–13.4 s baseline; 19.6–21.8 s saturated). Second, the replica wins every cell at every level—the mode

Table S13: Measured effective-bandwidth sweep on the calibrated platform (PCIe 4.0 x8 shared root; injector load L with achieved rates logged; n=3 warm reps, 4-bit, concurrency 1). Margins are $(T_{\text{staged}} - T_{\text{replica}})/T_{\text{replica}}$ at 8,192 and 16,384 tokens; τ_{eff} is the two-context slope $[D(16K) - D(8K)]/8{,}192$ with $D = T_{\text{staged}} - T_{\text{replica}}$; $\text{ctx}^*_{\text{eff}} = \Delta A/\tau_{\text{eff}}$. Values in parentheses are the second pass (l0 and sat re-measured in a separate session; see the drift note below).

Level	L (GB/s)	B_{eff}	margin @8K	margin @16K	τ_{eff} (ms/tok)	$\text{ctx}^*_{\text{eff}}$
l0	0	16.0	$+59.2\%$ ($+31.4\%$)	$+42.7\%$ ($+56.1\%$)	0.18	1,622
l6	5.98	11.6	$+29.9\%$	$+43.8\%$	0.52	545
l12	11.94	7.3	$+32.6\%$	$+34.2\%$	0.64	446
sat	21.97	$\to$0	$+35.9\%$ ($+30.1\%$)	$+37.9\%$ ($+58.1\%$)	0.64	444

decision is robust to link load, as the law predicts. An extended pass (`scripts/run_extended_bw_ladder.ps1`, `scripts/analyze_bandwidth_sweep_extended.py`) measured the full mid-range ladder—4,096 / 6,144 / 8,192 / 12,288 / 16,384 / 24,576 tokens in both modes, plus 32,768 staged—at 6 and 12 GB/s (n=3 warm reps each), for 24 additional cells: the replica wins all 24 (margins $+29.9\%$ to $+67.5\%$), so the conc-1 regime law (replica wins every feasible context) is invariant across four effective bandwidths and 28 total forced cells, not an artifact of the base link. Third, the two-context transport-tax slope is a weak estimator: over the extended 6-context window it is not monotonic in load (0.24–0.48 ms/token, with an unphysical free-fit ΔA), i.e., the per-token tax is below the measurement noise floor of this consumer platform at mid contexts. The $\tau \propto 1/B$ scaling's *clean* anchors are therefore the engine crossovers (llama.cpp $\leq 1{,}024$ vs. framework < 512 vs. vLLM $< 8{,}192$), not the contention sweep; the sweep's contribution is the *robustness* of the mode decision to bandwidth, not a calibration-grade $\tau(B)$ curve. The caveat: this consumer platform shows large between-pass variance ($\pm$20–40% on identical unloaded cells measured minutes apart; cf. the l0 parentheses above), so the intermediate throttle levels are reported as a qualitative direction check, not as calibration-grade $\tau(B)$ points. The saturated endpoint was reproduced across two passes and across the original contention suite (25.7 GB/s injector); the smaller split slowdown recorded there ($+9\%$/$+3\%$) reflects that suite's faster machine state, while the fresh passes show larger saturated slowdowns ($+48\%$–$+62\%$ staged at 8K, $+29\%$–$+86\%$ at 16K). All raw cells are in `reports/costmodel/contention/bw_sweep/` (pass 1 and rerun), the rate-limited injector is `scripts/pcie_contention_throttled.py`, and the analysis scripts are `scripts/analyze_bandwidth_sweep.py` (two-point) and `scripts/analyze_bandwidth_sweep_extended.py` (6-context regression).

Measured second physical link (BIOS Gen3 x8). The one remaining axis—a second physical bandwidth regime—is now measured rather than runbooked: a BIOS PCIe-generation downgrade of the same x8 slot caps the link at Gen3, and a pinned-copy probe measures the effective relay bandwidth at 6.65 GB/s (Gen3) versus 13.3 GB/s (Gen4), a clean $2\times$ reduction (`scripts/pcie_bandwidth_clean.py`). On *same-session* full ladders at both speeds (`scripts/run_gen3_ladder.ps1` / `run_gen4_ladder.ps1`, `scripts/compare_gen3_gen4.py`; ΔA and the quadratic term held at the calibration values so only τ varies), the transport tax rises from $\tau = 0.56$ ms/token (Gen4) to $\tau = 1.04$ ms/token (Gen3)—a ratio of $1.85\times$ against the $2.0\times$ predicted by $\tau \propto 1/B$—and the crossover shifts $\text{ctx}^* = 507 \to 274$ tokens. This is the second measured operating point of the law on a physical link, closing the "single physical topology" limitation: the $\tau \propto 1/B$ scaling and the $\text{ctx}^*(B)$ shift are now anchored by two same-session physical bandwidths, in addition to the engine, contention, and OS axes. (The same two GPUs and root complex are used, so this is a link-speed axis, not a different interconnect family; NVLink/P2P remains unmeasured; the wider x16 link is measured on a second host in the next paragraph.)

Measured third physical link and second host (PCIe 4.0 x16, cross-NUMA, 2×RTX 4090 24GB). The remaining runbooked axis—a wider link on a second physical host—is now measured with the identical protocol (same pinned-copy probe, same pinned 7B revision, same true-token ladder, n=7 warm reps per cell, reps $= 8$; all raw cells harvested under `reports/costmodel/cloud_harvest/`). The probe finds D2H 20.47 / H2D 21.82 / serial relay 17.62 GB/s—only $1.33\times$ the x8 relay, not the $2\times$ slot ratio: on this host the two cards sit on *different* NUMA nodes

(`nvidia-smi topo -m`: GPU0 NUMA3, GPU1 NUMA2, SYS), so the relay crosses the inter-socket UPI link, which bottlenecks below the slot spec. Effective relay bandwidth is therefore a property of the full topology *path*, not of the per-card slot—precisely the bandwidth-labeled-graph abstraction the selector consumes. Two decision-relevant results follow. (i) Regime-law invariance across hosts: the replica wins every feasible conc-1 context from 512 through 65,536 ($15/15$; margins $+21.1\%$ to $+93.1\%$, monotone in context from 8,192 ($+31.8\%$) to 65,536 ($+93.1\%$)), and replica-only cells complete at 98,304/131,072 tokens (split not attempted there by ladder design)—the replica-wins-conc-1 recommendation transfers across hosts and GPU generations; the calibrated estimator itself does not transfer (see below). On the same host, one 4090 alone serves 65,536 tokens in 32.7 s while the replica pair serves it in 16.9 s ($+48\%$)—the second GPU helps on long context precisely as a replica—and the base host's load axes reproduce: the decode-length invariance (replica wins 8K at 128/256/512 new tokens by $+26.9/+25.4/+22.8\%$) and the concurrency rule (the split wins 1,024/2,048 at c2 and c4, and 16,384 at c2; c4 16,384 is a near-tie). (ii) The transport tax at 17.6 GB/s is reported as a *bound*, not a point: the cross-host free fit degenerates like the base-host free fit (negative ΔA; on the cloud b_R additionally clamps to zero), so the split's excess linear slope $\tau \leq 0.187$ ms/token is an upper bound on the tax, and the held estimator does not transfer (per-context values turn negative beyond 12,288 tokens because the local quadratic compute term overestimates the 4090's compute). The bound sits $3\times$ below the held $\tau = 0.560$ ms/token at 13.3 GB/s—directionally consistent with $\tau \propto 1/B$, but the law's calibration-grade anchors remain the same-host Gen3/Gen4 pair (1.04/0.560, ratio $1.85\times$ vs. the $2.0\times$ predicted). The cross-host slope drop (local $b_S = 0.727$–0.745 vs. cloud 0.187 ms/token) shows the linear slope is compute-dominated on the slow host: the τ signal only separates from compute when compute terms are held at calibration (same host) or when compute shrinks to the clamp (fast host).

Model-size axis on the second host (14B, GQA-8, same pinned revision). Same protocol on the same two 4090s: core c1 ladder to 65,536, c2/c4 stability at 1,024/2,048/16,384, single-GPU 32K/65K, and an 8K decode matrix (128/256/512 new tokens), reps $= 8$ (n=7 warm). Four size-axis results. (i) Regime-law transfer: the replica wins all 13 feasible c1 contexts (512–32,768; margins $+20.3\%$ to $+58.3\%$, $+29.1\%$ at 8,192 vs. the 7B's $+31.8\%$), and the decode matrix is near-identical to the 7B's ($+26.7/+24.8/+21.5\%$ vs. $+26.9/+25.4/+22.8\%$)—the decision is size-invariant wherever both modes are feasible. (ii) The replica ceiling is size-bound exactly as the feasibility gate predicts: the 7B reaches 131,072 tokens; the 14B works at 32,768 and OOMs at 49,152/65,536, a bracket $(32{,}768, 49{,}152)$ that contains the KV-ratio prediction $131{,}072/3.43 = 38{,}229$ (the 14B's KV-per-token is $3.43\times$ the 7B's: 48 layers $\times$ 8 KV heads vs. 28 $\times$ 4). (iii) Single-GPU reach (24 GB): one 4090 serves 65,536 tokens in 75.3 s, and the replica pair beats it at 32,768 (18.8 vs. 29.9 s, $+37\%$); at 65,536 the split (75.6 s modeled) only matches the single card—on the 14B the second GPU's long-context benefit *is* the replica, and it ends where the gate predicts. (iv) The free fit again degenerates ($\Delta A = -957.5$ ms, negative ctx*), so cross-size τ is not identifiable: b_S rises $2.2\times$ (0.187→0.417 ms/token) against the $3.43\times$ KV ratio—the per-token slope is still compute-dominated even on the 4090. The clean size-axis evidence is the regime law, the ceiling bracket, and the decode matrix, not τ. The two-model $\times$ two-host transfer matrix consolidates the axis (Table S14); the 14B local x8 cell comes from the same-protocol local ladder (reps $= 6$, n=5 warm): the replica wins 7/7 feasible contexts (512–4,096, margins $+5.7\%$ to $+34.8\%$) and OOMs at 6,144, a ceiling bracket $(4{,}096, 6{,}144)$—$\sim 5\times$ below the 7B's $(24{,}576, 26{,}000)$ and steeper than the KV-ratio extrapolation ($24{,}576/3.43 = 7{,}165$, above the bracket): on the 12-GB host the NF4 weight share dominates the budget, while on the 24-GB host the KV term dominates and the ratio holds—both regimes are the same gate arithmetic (weights + KV + activations) with different dominant terms. The 14B split reaches 32,768 on 12 GB (OOM at 49,152), and one 3060 alone serves 8,192/12,288/16,384 in 28.5/43.8/61.0 s (n=5 warm each)—the size axis shrinks the replica's reach hardest on the small cards, as the gate predicts (weights + KV + activations against 12 GB).

SGLang axis (measured). A third engine point for the crossover law's engine ordering: SGLang 0.5.17 (Zheng et al., 2024) under WSL2 (torch 2.11.0+cu130, triton attention backend; FlashInfer's JIT path is unavailable without a CUDA 12 toolchain in this WSL image) serving Qwen2.5-7B-Instruct-AWQ on the same two cards, same true-token protocol:

On this link SGLang's tensor-parallel split beats both the single device and the data-parallel replica at 8K/16K—its per-token transport tax (well-overlapped activation collectives) is smaller than the HF relay's $\tau = 0.37$ ms/token and even llama.cpp's layer split, so its replica-vs-split crossover sits *above* 16,384: the engine ordering of crossovers—HF < 512, vLLM PP-2 $< 8{,}192$, llama.cpp

Table S14: Two-model × two-host transfer matrix (conc-1, NF4 4-bit, same pinned revisions on both hosts; warm n=7 cloud / n=5 local reps; distinct prompts). The 14B's KV-per-token is 3.43× the 7B's (48 layers × 8 KV heads vs. 28 × 4). "Wins" counts feasible c1 contexts won by the replica; ceilings are measured brackets (last working / first OOM context).

	7B (GQA-4)	14B (GQA-8)
x8 local, 13.3 GB/s, 2×12 GB	held $\tau = 0.560$ ms/tok; 14/14 wins; ceiling 24,576–26,000	free fit degenerate ($\tau = -0.04$ ms/tok; replica arm 512–4,096); 7/7 wins; ceiling $(4{,}096, 6{,}144)$
x16 cloud, 17.6 GB/s, 2×24 GB	free-fit bound $\tau \leq 0.187$ ms/tok; 15/15 wins; ceiling 131,072	free fit $\tau = 0.111$ ms/tok ($\Delta A < 0$); 13/13 wins; ceiling $(32{,}768, 49{,}152)$

Table S15: SGLang parallelism axis on the calibrated platform (WSL2, Qwen2.5-7B-Instruct-AWQ, 64 new tokens, warm n=3 per-request walls, ms). tp1 = single device; tp2 = tensor parallel over both cards; dp2 = data parallel (two replicas, round-robin).

Config	8,203 tokens	16,395 tokens
tp1 (single)	6,772 ± 71	9,096 ± 11
tp2 (tensor parallel)	4,212 ± 58	5,485 ± 8
dp2 (data parallel)	7,014 ± 250	9,180 ± 462

$\leq$ 1,024, SGLang TP $>$ 16,384—follows each engine's communication efficiency exactly as the law predicts, now with three engines and a data-parallel replica measured on the same hardware. The DP (replica) config matches single-device per-request latency while doubling serving capacity; SGLang's DP-attention variant (replica with cross-worker KV transfer) is not measured here (it requires the FlashInfer backend, whose JIT needs a CUDA 12 toolchain in this WSL image), and is flagged as the natural next engine point. Protocol and raw cells: `scripts/sglang_engine_axis.py`, `reports/sglang_axis/`.

SGLang under load, the DP-attention constraint, and the planner emission. Three follow-ups complete the SGLang axis. (i) Under the same staggered-arrival protocol as the llama.cpp tail study (12 requests, 64 new tokens, 7,936-token prompts, servers `--context-length 32768`), SGLang's TP-2 also wins tail latency and goodput: TTFT P95 3,339 ms and 2.89 req/s vs. 5,252 ms/1.88 req/s for tp1 and 5,341 ms/1.83 req/s for dp2 (`scripts/sglang_goodput_study.py`, `reports/sglang_goodput/`). (ii) SGLang 0.5.17's DP-attention variant asserts `tp_size % dp_size == 0` (tp2+dp2 needs four devices), so the cross-worker-KV replica is not measurable on this two-GPU host; the pure data-parallel dp2 config is the measured replica point, and DP attention is flagged as the next engine-axis point on a four-GPU machine. (iii) The planner's load-aware emission matches the measured optimum: at concurrency 1 it emits the replica (one single-GPU server), at concurrency $\geq$ 2 the staged-split server, and it rejects 16K×c4 on the feasibility gate (`build\_llamacpp\_plan`, CPU-only check)—the same regime the tail study measures, closing the emission–measurement loop with zero user decisions.

Tail latency under staggered arrivals (llama.cpp, measured). To complement the single-request and batch-wall evidence with per-request tail behavior, we run the planner-emitted llama.cpp configurations under staggered arrivals: 12 requests per config, dispatched with a deterministic 1.5–3.0 s spacing cycle, each streaming `/completion` (n_{predict}=64, temperature 0, `cache_prompt=false`, distinct nonce), per-request TTFT (first streamed chunk) and completion recorded client-side. Servers: `-c 32768`, `-np 4` (8K) / `-np 2` (16K) so each slot holds one full prompt plus its generation; prompts are built to exactly 7,936 / 16,128 tokens against llama.cpp's own tokenizer via the server's `/tokenize` endpoint (driver `scripts/run_tail_latency_study.py`, raw cells in `reports/tail_latency/`).

Under load the split wins tail latency and goodput at both contexts (TTFT P95 2.9/6.2 s vs. 4.5/10.1 s for the single device; goodput 3.0/1.54 vs. 2.1/1.05 req/s)—the same direction as the concurrency rule the selector encodes (replica at concurrency 1, split from concurrency 2), now evidenced in per-request tails rather than batch walls. Two caveats: single and replica are near-identical at these arrival rates because each server's four slots absorb the offered load before the second replica card

Table S16: Planner-emitted llama.cpp configs under staggered arrivals (Q4_K_M, 12 requests, 64 new tokens; TTFT in ms; goodput over the makespan).

Config	TTFT P50	TTFT P95	TTFT max	Goodput (req/s)
8K: single	4,307	4,447	4,538	2.11
8K: replica	4,415	4,575	4,586	2.09
8K: split	**2,820**	**2,899**	**2,933**	**3.00**
16K: single	9,895	10,057	10,203	1.05
16K: replica	9,537	10,092	10,328	1.04
16K: split	**6,039**	**6,179**	**6,646**	**1.54**

matters (the replica's value is memory-feasibility and higher offered load, not this arrival rate); and the arrivals are a deterministic stagger, not a Poisson process—an arrival-process study with SLO analysis remains future work.

Long-context quality: 32K vs. 65K perplexity (measured). To bound RoPE-scaling quality across the window extension, we re-run the paper's sliding-window perplexity protocol at a 32,768-token window (ExLlamaV3 EXL3-4.0bpw, stock RoPE, same corpus and scoring rules; windows $[0, 32{,}768)$ and $[16{,}384, 49{,}152)$, last 4,096 tokens scored): combined PPL **7.51** (windows 7.3752 / 7.6531). With the paper's 65K runs (stock 7.65; manual YaRN (Peng et al., 2024) 7.57), quality drifts $< 2\%$ across 32K→65K and the 65K dynamic-NTK config stays within 4.2% of manual YaRN—no cliff at the native window. A LongBench/LEval subset would add real-task evidence; its data host is unreachable from this network (the dataset loads via a script with Google-Drive-hosted files), so it remains flagged future work on an unrestricted network. Raw cells: `reports/ppl/exl_stock_32k.json` (script `scripts/ppl_exl.py`).

OS axis (measured, same-session interleaved). To test the sensitivity of the calibrated cost model to the OS/runtime stack, we re-measure the forced cells on the same machine under WSL2 (Ubuntu 24.04, torch 2.11.0+cu130, transformers 5.14.1, bitsandbytes 0.50, a WSL2 venv) interleaved back to back with the native Windows runs (.venv, torch 2.11.0+cu128, transformers 5.11.0), one pair per config to keep the between-pass drift from dominating the OS difference. Protocol identical (4-bit NF4 on the fly, distinct prompts, n=3 warm reps); HF weights are shared through the same on-disk cache.

Table S17: Cross-OS cells (same session, interleaved pairs; n=3 warm reps). Ratios are WSL2/native-Windows per-request warm means.

Cell	Win (ms)	WSL2 (ms)	Ratio	Decision
512, c1, replica	2,586	2,364	0.91	rp (WSL) / st (Win, noise band)
512, c1, staged	2,401	2,566	1.07	—
8,192, c1, replica	8,140	8,687	1.07	rp wins (both)
8,192, c1, staged	10,650	10,842	1.02	—
16,384, c1, replica	14,793	15,849	1.07	rp wins (both)
16,384, c1, staged	21,618	22,266	1.03	—
8,192, c4, replica	39,281	15,668	0.40	*diverges* (see text)
8,192, c4, staged	30,244	27,410	0.91	st wins (Win) / rp wins (WSL)

The decision layer's behavior transfers at concurrency 1: the WSL2/native latency ratio is 0.91–1.07 across 512/8,192/16,384 (per-request warm means, n=3), the replica wins every c1 cell at 8K/16K on both stacks, and at 512 both stacks sit inside the estimator's noise band (the verdict flips there—a live illustration of the band). At concurrency 4 the framework's *own batch path* diverges across stacks: the Windows replica batch wall (39.1–39.4 s warm) matches the 8/15 calibration cell (44.3 s process-cold), while WSL2 completes the same 4×64-token batch in 14.0–17.3 s (per-request 3.5–4.3 s vs. 9.8–9.9 s); the staged c4 cells stay comparable (27.4 vs. 30.2 s, ratio 0.91). Consequently the c4 decision flips across stacks (staged wins on Windows, replica on WSL2). We report this as a genuine portability finding—the mode decision's constants are stack-dependent for batched replicas, which is precisely the per-environment recalibration the selector's EWMA telemetry exists to absorb—and note that the framework's own vehicle is not the engine the integration layer targets at scale.

Table S18: Selector decisions across synthetic topologies (4-bit, concurrency 1; CPU-only planner sweep). rp = request-parallel replica. The crossover moves right with bandwidth; the replica ceiling is bandwidth-independent.

Topology	512	2,048	8,192	16,384	24,576	32,768
USB2 hub (0.06 GB/s)	rp	rp	rp	rp	rp	split
USB4/Thunderbolt (3 GB/s)	rp	rp	rp	rp	rp	split
PCIe 4.0 x8 (calibrated)	rp	rp	rp	rp	rp	split
PCIe 4.0 x16 (32 GB/s)	split	rp	rp	rp	rp	split
PCIe 5.0 x16 (64 GB/s)	split	split	rp	rp	rp	split
NVLink (200 GB/s)	split	split	rp	rp	rp	split

Root cause of the c4 divergence (measured). A direct diagnostic (`scripts/diag_rp_c4.py`) isolates the effect: on both stacks the replica workers are pinned one-per-card (pins $[0, 1]$), and concurrent per-worker batches overlap on both stacks. The difference is the framework vehicle's per-worker batch formation. On WSL2, one worker's 2-request batch takes 12.6–14.2 s (the 4-request batch 24.9 s, i.e., $\sim$6.2 s per request—genuine batching with shared prefill); on Windows the same 2-request batch takes 31.1–35.6 s ($\sim$17.8 s per request—near-sequential, no prefill sharing), and the concurrent 2×2 wall is 38.1 s. The Windows behavior reproduces the 8/15 calibration cell (44.3 s process-cold) and explains the c4 verdict flip: it is a defect in the framework's own HF vehicle (its batch path degenerates on Windows), *not* a decision-layer artifact, and it is the concrete reason the decision layer's constants must be recalibrated per environment—the EWMA's job. Fixing the vehicle's Windows batch path is recorded as future engineering work; the decision layer above it is unaffected. Raw cells: `reports/diagnostics/rp_c4_wsl.json` (the Windows c4 batch walls are recorded inside `reports/costmodel/wsl_os_axis/cross_os_summary.json`, as the separate Windows JSON from that run is not in the bundle); `reports/costmodel/wsl_os_axis/` (driver `scripts/run_cross_os_suite.ps1`, analysis `scripts/analyze_cross_os.py`).

Engine axis (measured). The same PCIe 4.0 x8 link, three engines: the HF runtime's per-layer relay ($\tau = 0.37$ ms/token) puts the crossover below 512 tokens; vLLM's pipeline-parallel-2 beats its single-GPU arm at 8,192 and 16,384 (warm totals 4,669 vs. 6,874 ms and 5,567 vs. 8,315 ms; $-32\%/-33\%$, crossover below 8,192); llama.cpp's layer split has its crossover in $\leq$ 1,024—a $\sim20\times$ spread across engines, ordered by communication efficiency as the law predicts. Protocol: vLLM 0.25.1, official AWQ artifact, `-max-model-len 32768`, warm n=3, identical true-token prompts (`scripts/wsl_vllm_engine_crossover.sh`, `reports/baseline_vllm/engine_crossover_{pp2,single}.json`); note this in-window arm differs from the YaRN-override arm of Table S38.

Contention axis (measured). A PCIe-saturating background load (25.7 GB/s aggregate pinned-copy traffic, `scripts/pcie_contention_load.py`) slows the split by $+9\%/+3\%$ at 8,192/16,384 and the replica by $+19\%/+19\%$: the single-card replica concentrates per-card copy interference, but the mode decision holds (margins $-42\% \rightarrow -37\%$, $-46.6\% \rightarrow -38.4\%$). The percentages are anchored to the contention run's same-session baseline (split 13,171/26,145, replica 7,585/13,969 ms); a later session re-measured the 16,384 baseline cells at 20,741/17,241 ms (known 4-bit bimodality), which keeps the 8K percentages unchanged and the 16K percentages direction-only—the decision holds under either baseline. Per-cell records in `reports/costmodel/contention/`.

Model axis (measured). Llama's per-model KV-slope factor ($1.91\times$ vs. Qwen's $3.424\times$) transfers the feasibility law across models (Section 6.7).

Synthetic sweep (planner-level, disclosed). The seven-topology selector sweep (the table below):

Randomized off-grid decision set. Twelve seeded-random (context, concurrency, generation) cells measured in both modes with generation-forced prompts (`scripts/run_random_grid_suite.py`, seed 20260815); prediction vs. warm-mean hindsight scored in `reports/costmodel/grid_random/summary.json`. Result: **11/12 exact, 12/12 with a 1% tie margin**—the single miss is a 20 ms tie at 6,144$\times$concurrency-6 (staged 40,388 vs. replica 40,368 ms)—and five of the twelve cells land in the both-modes-infeasible regime, which

the feasibility gate rejects correctly in all five. A second, independent 12-cell set (seed 20260818, n=5 reps) scores **11/12 exact**; its single miss is the shortest cell (768×concurrency-1, generation 64), where the replica beats the predicted split by 19%— the crossover neighborhood, the selector's least-identified region (Appendix 5.2). Combined: **22/24 exact** across 24 randomized off-grid cells, with four further both-modes-infeasible cells correctly rejected in the second set.

Throughput vs. concurrency at 8,192 (measured). Warm batch walls (replica/split, ms): c1 7,585/13,296, c2 30,911/18,985, c4 37,920/30,227, c8 OOM/60,514. Throughput (req/s): the replica leads at c1 (0.132 vs. 0.075), the split overtakes from c2 (0.105 vs. 0.065 at c2; 0.132 vs. 0.105 at c4) and is the only feasible mode at c8— the throughput crossover at concurrency 2 matches the selector's concurrency rule (Section 6.4, mechanism 2).

NoLiMa subsample (≤32K, measured). To complement the literal-match RULER/NIAH probes (Hsieh et al., 2024; Kamradt, 2023), we run the NoLiMa benchmark (Modarressi et al., 2025), whose needles have minimal lexical overlap with their questions, on the framework's serving stack (vLLM 0.25.1, AWQ, greedy, default task template, `-max-model-len 32768`, `--enforce-eager`): a subsampled sweep of 5 needle experiments × 26 depth placements × lengths 1K/2K/4K/8K/16K, with 13 placements at 32K (mirroring NoLiMa's own reduced long-length placements) (`scripts/wsl_run_nolima.sh`, `scripts/aggregate_nolima.py`, `reports/nolima/summary.json`). Accuracy by length: **1K 43.8%, 2K 36.1%, 4K 27.6%, 8K 19.2%, 16K 14.1%, 32K 4.2%**—9.5% of the 1K base at 32K. The degradation-without-literal-matches pattern matches NoLiMa's own finding for 7–8B models; our absolute scores sit below the NoLiMa paper's Llama-3.1-8B run, which we attribute to the serving protocol (AWQ on a consumer card, default template, empty system prompt) and disclose. The relevant comparison for the framework's *usable-context* claim is therefore the *relative* degradation against the 1K base (9.5% at 32K), reported alongside the 42/42 and 30/30 literal-retrieval results rather than as a standalone capability claim.

Reproducibility statement (full). All measurements reproduce from the committed scripts and per-rep JSON artifacts under `reports/`: selector evaluation and calibration (`eval_cost_model_accuracy.py`, `calibrate_selector_latency_model.py`, `verify_mode_crossover_law.py`), cell/RULER/contention runners, and the planner-driven llama.cpp demo (`run_llamacpp_planner_demo.py`). Protocols (token-verified prompts, cold/warm taxes, generation matching) are in this supplement. Hardware: two RTX 3060 12 GB (PCIe 4.0 x8), Ryzen 9 9950X3D, Windows 11 + WSL2; PyTorch 2.11, Transformers 5.11.0, llama.cpp b10066, ExLlamaV3 1.1.0, vLLM 0.25.1.

5.8 Engine integration protocol

Planner-driven llama.cpp (`src/tworope/integration/ llamacpp_controller.py`, `scripts/run_llamacpp_planner_demo.py`): replica = one llama-cli process per worker GPU (CUDA_VISIBLE_DEVICES), split = one dual-GPU process with `-ts 1,1`; slot = $\min(\text{ctx} + 1024, 32{,}640)$; identical prompt protocol to the hand-tuned ladder (`reports/baseline_llamacpp/ladder.json`). Measured walls: planner replica 8.37/13.84 s at 8,192/16,384 vs. hand-tuned split 6.44/10.30 s; forced 12,288: replica 11.36 vs. split 8.49 s (the engine-specific crossover bracket); planner split 17.36 s at 32,000. Barriers recorded: the official server's 32,640 admission clamp, llama.cpp's KV ownership (the planner only sizes and selects), a /comment template-parsing 500 on raw word-fill output, and build b10066's `-v` verbosity bug.

6 ExLlamaV3 baseline protocol details and erratum

Main paper, Section 6.3 positions the framework against llama.cpp and ExLlamaV3. This appendix records the ExLlamaV3 protocol in full, including the needle-arm erratum (Appendix 6.2) and the perplexity protocol that settles the ±YaRN question. Source bundle: `reports/baseline_exllama/`, `reports/ppl/`, `reports/needle/` (runners `scripts/run_exl_probe.py`, `scripts/ppl_exl.py`, `scripts/run_needle_sweep.py`).

Table S19: ExLlamaV3 tensor-parallel speed ladder (dual RTX 3060). Cold rep 0 adds ∼30–40% (31.4 / 41.6 / 80.3 s respectively). Generation stopped at EOS at 62–63 tokens, close enough to 64 for the ladder comparison. Outputs are degenerate repeated filler by design (latency-only cells).

Context (actual)	warm (ms, n=2)	prefill (tok/s)	GPU0 peak (MiB)
8,203 (8,214)	22,763	∼361	2,383
16,395 (16,407)	32,416	∼506	2,623
32,779 (32,791)	76,226	∼430	3,056

6.1 Environment and speed protocol

- **Runtime:** ExLlamaV3 1.1.0 precompiled wheel (`+cu128.torch2.11.0-cp312-win_amd64`) in an isolated virtual environment (the framework's own environment untouched); torch 2.11.0+cu128; triton-windows 3.7.1 (required — the Triton paged-attention kernels are the default backend); the native tensor-parallel backend.
- **Model:** official EXL3 pre-quantized artifact (`turboderp/Qwen2.5-7B-Instruct-exl3`, branch `4.0bpw`, by the ExLlama author; 4.77 GB).
- **Speed protocol:** aligned with the framework's cells — true-token prompts (HF-tokenizer counted), a distinct nonce prompt per repetition, greedy decoding, max_new_tokens $= 64$, $n = 3$ (warm = mean of reps 1–2).
- **Memory caveat:** the native tensor-parallel backend spawns worker processes; `torch.cuda.max_memory_allocated` in the main process only sees its own device (GPU0), so GPU1 reads 0 artifactually. The split is symmetric, so GPU0's peak is the representative per-GPU figure.

Table S19 gives the measured speed ladder.

Head-to-head (Table S19, against the derived llama.cpp normalization of Table S38): ExLlamaV3 TP is 3.2–4.4× slower than llama.cpp and 1.2–1.7× slower than the framework's staged-split at every comparable context (2.3× slower than the framework's best measured mode at 16K) — the per-layer all-reduce over PCIe 4.0 x8 without P2P dominates (measured prefill ∼360–510 vs. llama.cpp's ∼2,000–2,300 tok/s on the same two cards). ExLlamaV3 is therefore a *weaker* speed baseline than llama.cpp at ≤32K on this platform, not a stronger one; where it changes the picture is admission beyond 32K (below).

6.2 Erratum: both needle arms in fact executed stock RoPE

The over-window needle probe ran two intended arms at 65,536 tokens: *stock* (untouched config) and *manual YaRN* (`rope_scaling={rope_type:yarn,factor:2.0,original_max_position_embeddings:32768}` injected into the config before model load). **Erratum (this appendix; the main paper's Section 6.3 positions this baseline but does not discuss the erratum):** ExLlamaV3 parses `rope_scaling` into a frozen `RopeSettings` object inside `Config.from_directory`; editing `config.config_dict` *after* `from_directory` — as the probe runner did — is a silent no-op. The arm labeled "manual YaRN" in the needle series therefore actually executed **stock RoPE**, and the needle series bounds the YaRN-necessity question from the stock side only. The "no-YaRN necessity" conclusion is unchanged (both arms were stock), but the "YaRN" labels in those needle records should be read as stock-RoPE replicates. The corrected injection (patching `config.rope_settings` directly, verified: YaRN perplexity now differs from stock) is used in the perplexity protocol of Appendix 6.3.

For the record, Table S20 lists both needle arms (needle Aurora/7481, depths 10/50/90%, greedy, max_new_tokens $= 32$, $n = 1$ per cell).

Re-run with the corrected injection. Because the erratum left the YaRN side of the question measured only by *stock* replicates, the 65,536-token needle arm was re-run with the corrected injection (`scripts/run_exl_probe.py--yarn`). Two independent checks show the fix is not another silent no-op. *Structural:* the runner now builds the RoPE object for both configurations and compares the table it will actually use (`reports/baseline_exllama/rope_check_65k.json`);

Table S20: The 65,536-token needle arms. Left block: uniform-filler haystack. Right block: natural-text haystack (47,592 tokens of real prose tiled once to 65,535–65,536; prompt files shared identically across all engine configurations). All cells retrieved the needle verbatim (12/12 across the natural-haystack block). The last ExLlamaV3 row is the re-run with the corrected YaRN injection (Section 6.2); the rows above it are the originals, whose two "arms" in fact both executed stock RoPE. The 84.7 s and 80.1 s depth-90 natural cells are a warm Triton autotune-cache artifact (cache persisted across processes), not a YaRN speed effect.

	Uniform-filler haystack			Natural-text haystack		
Depth	10%	50%	90%	10%	50%	90%
ExLlamaV3, stock RoPE	✓ 161.8 s	✓ 158.1 s	✓ 155.3 s	✓ 136.3 s	✓ 137.3 s	✓ 163.5 s
ExLlamaV3, "YaRN" (= stock)	✓ 143.4 s	✓ 162.2 s	✓ 141.3 s	✓ 136.7 s	✓ 148.6 s	✓ 84.7 s
ExLlamaV3, YaRN (corrected)	✓ 152.1 s	✓ 139.8 s	✓ 79.3 s	✓ 145.0 s	✓ 133.6 s	✓ 80.1 s
tworope staged (auto NTK)	(uniform series: Appendix 30)			✓ 222.4 s	✓ 227.4 s	✓ 231.2 s

the stock and YaRN inverse-frequency tables hash to `f047d45a` and `9aaaa682`, and the attention factor moves $1.0 \rightarrow 1.0693$. *Behavioural:* the two original arms produced near-verbatim identical continuations—independent corroboration that both ran stock RoPE—whereas the corrected YaRN arm's continuations differ from the stock ones from the first sentence after the answer. Retrieval is nevertheless unchanged: all six re-run cells return the needle verbatim (3/3 uniform-filler at $152.1/139.8/79.3$ s, 3/3 natural-text at $145.0/133.6/80.1$ s). The conclusion therefore no longer rests on an inference from two stock runs: at $65{,}536$ tokens this engine retrieves the needle with *and* without YaRN, so YaRN is not what carries the reach—the engine's stock window plus its own position handling already does. The 79–80 s depth-90 cells reproduce the warm-autotune artifact noted in the caption, now in a second, independent run.

Every arm's answers converge to the same greedy continuation (same weights, same prompt), and ExLlamaV3's no-YaRN answers are near-verbatim identical to its YaRN-labeled answers. On the natural haystack the manual-expert path is $\sim 1.5\times$ *faster* than the framework at 65K (136–164 s vs. 222–231 s cold, one fresh process per repetition; the warm-ladder figure for the same context is 180.0 s, Table S38) — the inverse of the $\leq$32K ordering — the measured premium the framework pays for full automation (main paper, Section 6.3).

6.3 Perplexity protocol: Pride and Prejudice, two windows, 8,192 scored tokens

The residual risk to the necessity conclusion was long-form quality: needle retrieval is extractive, so a perplexity delta at 65K $\pm$YaRN was measured. Protocol (as corrected after the erratum above):

- **Corpus:** Project Gutenberg #1342 (*Pride and Prejudice*, headers stripped), 171,751 tokens.
- **Windows:** two 65,536-token windows, A $= [0, 65536)$ and B $= [16384, 81920)$. Per window, the **last 4,096 tokens are scored** (RoPE positions 61,440–65,535, entirely in the beyond-32K extrapolation region); combined across both windows, **8,192 scored tokens**. NLL in chunked fp32 log-softmax.
- **Forward scheme:** both engines use chunked KV-cache prefill (8–16K chunks) plus one final forward over the last $S{+}1$ tokens; a single 65K forward OOMs on 12 GB cards in both engines. ExLlamaV3 path: contiguous non-paged cache (identity block table). Framework path: HF `DynamicCache` with per-chunk disk checkpointing for slice resumption.
- **Configurations:** *stock* — EXL3-4.0bpw, untouched RoPE; *YaRN* — same artifact with correctly injected `rope_scaling` (factor 2.0, original window 32,768); *tworope* — HF 4-bit NF4 + double quant, bf16 compute, sdpa, `rope_type=dynamic, factor=2.0` (the framework's automatic dynamic NTK).

Table S21 gives the measured perplexities.

Findings (main paper, Section 6.3): (1) stock RoPE does not collapse at $2\times$ trained length on natural prose (combined 7.65; Qwen2.5's `rope_theta` $= 10^6$ keeps $2\times$ extrapolation in-distribution for

Table S21: 65K sliding-window perplexity (lower = better). Peak VRAM: ExLlamaV3 9.4 GiB (layer-split lands on cuda:0); framework 4.3/9.2 GiB across both cards. Source: `reports/ppl/` (2026-07-19).

Configuration	window A	window B	combined (8,192 tok)	fwd ms/window
ExLlamaV3 stock RoPE	8.0579	7.2566	**7.6468**	$\sim$103,000
ExLlamaV3 + YaRN f=2.0	7.9031	7.2579	**7.5736**	$\sim$103,000
tworope dynamic NTK f=2.0	8.2754	7.5323	**7.8951**	$\sim$171,000

dense natural text), and hand-tuned YaRN recovers only $\sim$1% (7.57, -0.07); (2) the framework's automatic dynamic NTK stays within 4.2% of hand-tuned YaRN (7.895 vs. 7.574, $\Delta = +0.32$) and within 3.2% of stock RoPE, with zero per-model rope configuration; (3) combined with the 65K needle results (9/9 retrieval, Appendix 6.2), the necessity claim is bounded: at 2$\times$ context neither YaRN nor NTK is strictly necessary for this model on natural text — position-interpolation schemes are a robustness margin, not a necessity, at 2$\times$ for Qwen2.5-class high-θ RoPE models.

7 SCBENCH MULTI-TURN TASK EVALUATION

LongBench's data host is unreachable from this network, so downstream long-context task quality is measured on **SCBench** (Li et al., 2024), whose data (`microsoft/SCBench`, Hugging Face) is downloadable; we use the English *choice* and *QA* tasks (semantic retrieval over book-length contexts) with the official multi-turn protocol.

Protocol (replicates the authors' `GreedySearch` multi-turn mode, `github.com/microsoft/MInference/scbench`): raw prompts (no chat template; the official default), the official templates verbatim (including the "The the correct answer is" suffix), *golden previous answers* appended in follow-up turns (the official `pre_ans` mechanism), greedy decoding with `max_new_tokens` = 40, context middle-truncated to the window budget (official `manner="middle"`; 32,512/65,280 tokens for the 32K/65K windows, leaving room for the turn transcript), and the official scoring (both tasks boolean and per-turn: choice by the predicted option letter, QA by gold-answer containment). Engine: ExLlamaV3 4.0bpw Qwen2.5-7B-Instruct, stock RoPE at 32K and YaRN factors 2.0/4.0 at 65K/128K (128K additionally tensor-parallel across both cards under an 11.5-GB per-card cap; the paper's reference long-context engine, same recipe as the perplexity parity and the engine axis); 20 first-in-order samples per task and window (58 choice / 69 QA available; sample ids recorded in `reports/scbench/*.json`; the cross-engine OS-invariance check uses the first 10). The framework's own HF vehicle prefills 65K in $\sim$180 s, $\sim$1.2$\times$ slower than ExLlamaV3, so the engine-level run is used; multi-turn-over-one-context is exactly the workload the framework's prefix reuse already exploits (main paper, Section 4).

Metric note (revision). Earlier drafts of this appendix scored these two tasks with a partial-credit pair—`string_match_all` for choice (half credit if *either* the answer text or the option letter appears anywhere in the prediction) and normalized token F1 for QA. The official harness does not use those for these tasks: it maps `scbench_choice_eng` to `longbook_choice_eng` and `scbench_qa_eng` to `longdialogue_qa_eng` (`compute_scores.py` lines 417–419), both of which are boolean per-turn scores averaged over turns. Re-scoring the *stored* predictions with the official scorers—reproduced verbatim in `scripts/scbench_official_rescore.py`, output in `reports/scbench/official_rescore.json`—moves the numbers in both directions: choice falls ($0.775/0.794/0.819 \rightarrow 0.600/0.588/0.650$), because the old metric credited a letter that merely appeared somewhere in the prediction, and QA rises ($0.051/0.053/0.064$ F1 $\rightarrow$ $0.110/0.130/0.210$ accuracy), because token F1 under-credits the short quoted answers the prompt asks for. Everything below is stated on the official metrics.

Results (Table S22). On the objective multi-choice task accuracy is *flat* across an eight-fold window range: 0.600 / 0.588 / 0.650 at 32K / 65K / 128K and 0.600 at 256K (five-sample probe), against a 0.25 chance level for four-way choice and with no monotone trend—the 32K-to-128K gap is inside the ± 0.11 1σ band of an 80-turn mean. On free-form QA the official containment accuracy *rises* with the window: 0.110 / 0.130 / 0.210 at 32K/65K/128K. Together with the perplexity parity (Appendix 6.3), this is the task-level evidence that the reach the decision layer enables is usable:

Table S22: SCBench multi-turn results (ExLlamaV3 4.0bpw, Qwen2.5-7B, official protocol, 20 samples per task and window (5 at 256K); means over turns). 32K/65K: single GPU, stock / YaRN-2.0; 128K/256K: TP-2, YaRN-4.0 / YaRN-8.0. Both rows use the *official* metric for the task, re-derived from the stored predictions: the predicted option letter for choice, and gold-answer containment for QA (see the metric note below).

Task (metric)	Window	Turns	Mean	Per-turn means
Choice (accuracy)	32K	80	0.6000	0.70 / 0.55 / 0.45 / 0.70
Choice (accuracy)	65K	80	0.5875	0.60 / 0.65 / 0.45 / 0.65
Choice (accuracy)	128K	80	0.6500	0.65 / 0.60 / 0.50 / 0.85
Choice (accuracy)	256K	20	0.6000	0.60 / 0.60 / 0.60 / 0.60
QA (accuracy)	32K	100	0.1100	0.05 / 0.05 / 0.20 / 0.25 / 0.00
QA (accuracy)	65K	100	0.1300	0.15 / 0.05 / 0.15 / 0.20 / 0.10
QA (accuracy)	128K	100	0.2100	0.15 / 0.10 / 0.25 / 0.30 / 0.25

quality does not degrade out to 128K on either task, choice accuracy holds out to 256K where the platform's KV budget binds (the YaRN-8.0 extrapolation, a measured 10,391-MiB binding peak and roughly halved decode throughput buy no further accuracy), and generative quality improves with the wider window rather than collapsing.

The 128K rows exercise the second measured operating point of the per-model feasibility gate: Qwen2.5-7B (GQA-4, 57,344 B/token) KV at 128K is 7.00 GiB, which the TP-2 split holds at $\sim$3.5 GiB per card; the TP-2 run measured a 6,807 MiB binding-GPU peak under an 11.5 GB per-card cap. Llama-3-8B (GQA-8, 131,072 B/token) KV at 128K is 16.0 GiB—infeasible on two 12-GB cards even split—so 128K is Qwen-only, exactly as the per-model KV-slope gate predicts (main paper, Section 6.5).

OS and engine invariance (65K). The 65K window was replicated Linux-natively on vLLM 0.25.1 (the paper's Linux-native engine; Qwen2.5-7B-Instruct-AWQ, YaRN-2.0, PP-2, prefix caching, official multi-turn flow) against the Windows ExLlamaV3 run. On the same first ten samples and the official metrics: choice 0.575 (ExLlamaV3/Windows) vs. 0.500 (vLLM/WSL2), QA 0.080 vs. 0.080—the multi-turn task metrics agree across OS and engine to within a 10-sample band, so the 65K window's downstream quality is not an artifact of either environment. (ExLlamaV3's full 20-sample 65K means are 0.588 choice and 0.130 QA. Disclosure: the Windows ExLlamaV3 runner does not load in WSL2—its locally compiled CUDA extension import hangs—so the cross-OS pair covers both axes at once; this is a toolchain note, not a quality finding.)

Is the remaining level caused by our quantization? (measured control). Two independent checks say it is not. (i) *Published reference:* SCBench's own Table 11 reports multi-turn `En.QA` and `En.MC` for full-precision 8–9B models run in bfloat16 on A100s, and the task is hard *there* too—`En.QA` 29.0/29.3/33.6 and `En.MC` 53.1/57.0/50.2 for Llama-3-8B, Llama-3.1-8B and GLM-4-9B-1M (Li et al., 2024). Our 4-bit 7B sits *at or above* that band on choice (0.59–0.65 vs. 0.50–0.57) and below it on QA (0.11–0.21 vs. 0.29–0.34, across a different model family, size and window protocol): the QA shortfall is real, but the benchmark's own ceiling is the dominant term. (ii) *Own control:* we reran the same screen on the SGLang engine for the same model family at two precisions—AWQ 4-bit and full-precision weights—through the same harness, on the same twenty samples, with the official scorer, at *both* windows (Table S23). At 32K the QA arms are indistinguishable and the 4-bit arm is nominally *higher* (0.100 vs. 0.080), so the QA level is not a quantization artifact; choice carries the paper's only sizeable gap (0.475 vs. 0.588). The 65K row is the new measurement, and it answers the question this control exists for: **the gap does not compound with context**. On choice it *shrinks* from eleven points to four (0.512 vs. 0.550), and on QA it changes sign (+2 points at 32K, −5 at 65K). With n=80 choice turns the intervals still overlap at both windows, so the honest reading is "no detectable degradation that grows with context", not "quantization is free". Two engine facts are worth recording because both were silent traps: a request beyond the model's native 32,768 window is refused unless `SGLANG_ALLOW_OVERWRITE_LONGER_CONTEXT_LEN` is set, and the YaRN-2.0 recipe of the grid's 65K rows is injected through the engine's config override and then *read back* from the resolved `rope_scaling` (so it cannot repeat the silent no-op of Appendix 6.2).

Records: `reports/scbench/hp/`, `scripts/scbench_sglang_hp.py`, `wsl_run_scbench_hp.sh`. (*Serving detail:* SGLang's tensor-parallel path routes bfloat16 logits through

Table S23: Quantization control at the grid's own scale: SCBench multi-turn accuracy under the official metrics, same harness (SGLang, tp2), same twenty samples and scorer at both windows; Wilson 95% CI in parentheses. The 65K arm uses the grid's YaRN-2.0 recipe. The full-precision arm is served as float16 (text).

Window	Metric	AWQ 4-bit	Full precision
32K	Choice	0.475 (0.37–0.58)	0.588 (0.48–0.69)
32K	QA	0.100 (0.06–0.17)	0.080 (0.04–0.15)
65K	Choice	0.512 (0.40–0.62)	0.550 (0.44–0.65)
65K	QA	0.100 (0.06–0.17)	0.150 (0.09–0.23)

Table S24: SGLang continuous batching under a concurrent heterogeneous mix (8 requests, 64 new tokens each; Linux-native). dp2 = request-parallel replica.

Config	TTFT P95 (s)	TPOT P95 (ms)	Goodput (req/s)
tp1 (single)	44.6	725	0.172
tp2 (split)	26.6	435	0.288
dp2 (replica)	**23.7**	**393**	**0.318**

a symmetric-memory all-gather (`triton_symm_mem_ag.py`), which rendezvous-faults with SIGFPE on a PCIe-only host and offers no flag to disable it, so the full-precision arm is served as float16—equally "not 4-bit" for the question this control asks.)

Limitations disclosed: 20/58 and 20/69 sample subsamples (10 for the cross-engine OS-invariance check); a single engine (the paper's reference long-context engine); choice accuracy sits at 0.59–0.65 rather than saturating (a 7B-class 4-bit model on book-length multiple choice), and free-form QA accuracy is 0.11–0.21, so the QA evidence is the *trend* across windows rather than an absolute quality claim—the transcript-imitation artifact of the official golden-answer protocol caps what any model can score there. The precision control above covers 32K and 65K and shows no degradation that grows with context; it also shows the published full-precision 8–9B levels on these tasks (0.29–0.34 QA, 0.50–0.57 choice). The 128K row has *no* full-precision reference on this hardware, and that is arithmetic rather than omission: fp16 weights take 7.10 GiB per card and a 128K KV a further 3.50 GiB—88% of a 12 GB card before activations—while at `mem_fraction_static` $= 0.80$ the KV pool admitted at most 54,982 tokens even at 65K, which is why the 65K arm had to be raised to 0.86.

8 Continuous-batching head-to-head (SGLang, Linux-native)

To measure the mode decision under *continuous batching* with heterogeneous contexts on Linux, we fire eight requests *concurrently* (two each at 2,048 / 4,096 / 8,192 / 16,384 true tokens, 64 new tokens, temperature 0) at one SGLang 0.5.17 server per configuration— single-device (tp1), staged-split (tp2), and request-parallel replica (dp2, two data-parallel instances)— and record per-request TTFT, completion, and goodput (`scripts/sglang_cb_routing.py`; `reports/sglang_cb/*.json`). This is the continuous-batching counterpart of the Poisson-arrival simulation (Appendix 15) and the staggered-arrival llama.cpp study.

The request-parallel replica wins at *every* measured context (2,048–16,384): its two data-parallel instances absorb the concurrent arrivals without the single-device's queueing (tp1) or the split's all-reduce-per-token overhead (tp2), and its per-request latency is the lowest at all four contexts. This is exactly the concurrency rule the selector encodes—replica at concurrency ≥ 2 up to its per-card replica ceiling—and it is the empirical counterpart of the Poisson-arrival simulation's per-context routing verdict. The split's win is reserved for (a) contexts beyond the replica's per-card KV ceiling (32K+, engine axis) and (b) engines with a low split tax, where the staggered-arrival llama.cpp study shows the split winning 8K/16K (main paper, Section 6.9). Honest scope: a single 8-request burst, not a long multi-tenant trace; all contexts are below the replica's $\sim$26K replica ceiling, so the split-vs-replica routing bite is not exercised here.

Table S25: Mechanism ablation at 512 true tokens (the `tab:ablation` table inlined as prose in the main paper). rep 0 carries the first-at-length reconfiguration tax (Appendix 30) and is not comparable across rows; warm repetitions (mean ± std over reps 1–2, with the two warm values in parentheses) are the comparable numbers.

Configuration	rep0 (ms)	warm reps, mean ± std (ms)	restore fires?	Δ warm vs. full
full	33,894	4,057 ± 84 (4,116 / 3,997)	yes	—
no_reuse	33,650	4,640 ± 149 (4,746 / 4,534)	no (disabled)	+583 ms (+14.4%)
no_hierarchy	33,819	4,055 ± 86 (4,116 / 3,994)	yes	−2 ms (≈0%)
no_overlap	36,586	4,151 ± 697 (4,644 / 3,658)	yes	+94 ms (+2.3%, noisy)

9 THE ABLATION TABLE INLINED IN THE MAIN PAPER

For page budget the main paper (Section 6.2) presents the mechanism ablation as prose; the original `tab:ablation` table is restored here. Protocol: 512-true-token prompt (tokenizer-verified), bf16 staged runtime, max_new_tokens $= 64$, temperature $= 0$, $n = 3$ repetitions per configuration, and the *identical* prompt across repetitions and configurations — so all rows share the same repeat-prompt structure (prefix reuse is possible-but-disabled in `no_reuse` and enabled elsewhere). One process per configuration (toggle flags bind at plan creation); toggle application verified via `plan.disabled_features` containing `ablation:<name>`, and `no_reuse` additionally verified behaviorally (the prefix-store lookup counter never increments, and warm latency ≈ cold-prefill latency). Source bundle: `reports/ablation/` (runner `scripts/run_ablation_matrix.py`). Table S25 is the original table.

Reading (as stated in the main paper): prefix reuse is the only toggle with a measurable effect at this length — disabling it costs +583 ms per call (+14.4%) because every repetition re-pays the prefill that restore would have eliminated; this cross-validates the cold−warm delta of 545 ms at 512 tokens in Table S44 (the two deltas agree within noise).

The same toggles at long context. The overlap toggle is flat at 512 true tokens (+2.3%, within noise) and decisive once the context is long enough for prefill chunking to matter. Re-run at 8,192/16,384/32,768 true tokens with a distinct prompt per repetition (bf16 staged, $n = 2$ warm; raw data and protocol in `reports/ablation/longctx_*.json` and `reports/ablation/SUMMARY_LONGCTX.md`), disabling it *lowers* warm latency by 18.5% and 29.2% — but *raises* peak per-GPU allocation from 7,700/8,041 to 8,620/9,961 MiB (+12.0%/ + 23.9%) and makes 32,768 tokens infeasible (OOM before a single repetition completes). Chunked prefill is therefore a reach-for-latency trade rather than a latency optimisation — the same shape as the staged split's transport-for-reach trade — and a contribution this 512-token table could not have shown. Quantization matters here: these runs are bf16, whereas Table S42 is the 4-bit matrix, in which the runtime re-forces the chunk size on ≥2 GPUs and neutralises this toggle.

The `no_hierarchy` row differs: it is flat at *every* length because the runtime does not branch on it. The toggle sets `hierarchy_enabled`, which has no reader in the runtime, and `kv_policy`, which is already `standard` below 32,768 and whose only consumer above that emits a warning that tiered KV falls back to standard unless a dedicated backend is added. Measured at the three long-context lengths, its peak allocation is *bit-identical* to `full` (7,699.6/7,622.1, 8,041.1/7,963.8, 8,723.8/8,647.3 MiB) while warm latency varies by +4.4%/ + 2.9%/ − 0.6% — so that spread is a usable estimate of the between-process noise floor (≈ ± 4%), which is what makes the chunked-prefill effect above unambiguous. We report this row as inert rather than as a long-context null result.

Recompute. This matrix has no recompute row, and the primitive is not exercised by the workloads evaluated in Section 6. It has since been executed end to end and measured separately; see `reports/recompute_exec/` (`FINDINGS.md`, `tables.md`) in the released bundle.

10 METHOD DETAILS: STACK-ROOFLINE PROPERTIES, THE GREEDY SELECTOR, AND BANDIT SELECTORS

From the five-tuple workload to the deployed score. The tuple $w = (N, C, R, \phi, q)$ enters the deployed decision through two scalars: the per-request context length ctx (from C) and a concurrency term (from R); q only gates *admissibility* through the compressed tier's quality–cost trade-off, and N with ϕ set the per-model compute constant A_m. That is why Eq. (2) is one-dimensional in ctx rather than a function of all five components: the four dimensions of SE above are the explanatory summary, not the runtime score: in SE, each γ_d is the fractional gain over the strongest single-device baseline on dimension d, so the four are commensurable with $\lambda V_{\text{visible}}$. Eq. (2) prices a cold-prefill request; a warm-tier prefix hit replaces the prefill term with the measured restore cost, which is why the main paper reports reuse speedups separately from the mode decision.

10.1 STACK-ROOFLINE DERIVED PROPERTIES

These three properties complete the stack-roofline decomposition of the main paper (Section 3), whose per-step latency model is $T_{\text{step}} = \max(T_{\text{compute}}, T^{\text{visible}}_{\text{transport}}, T_{\text{kv-miss}}, T_{\text{sync}})$.

Communication hiding. If each transfer chunk completes before the consumer needs it, $T^{\text{visible}}_{\text{transport}} = 0$. This is the runtime's target behavior on narrow-interconnect steps. The feasibility of hiding depends on the ratio of transfer time to compute time per chunk, which is a function of the topology path $P_{u,v}$.

Throughput scaling. Under bounded metadata overhead and bounded placement imbalance, sustainable context capacity and throughput scale at least near-linearly with the number of devices, minus a throughput penalty proportional to the maximum path latency $L_{\max} = \max_{P_{u,v}} L_{P_{u,v}}$: path latency bounds the critical path per step, not the memory footprint.

Recovery control. If fetch and recompute costs are estimated with bounded error and decisions use hysteresis, online recovery cost stays close to the offline choice up to an additive error term proportional to the topology's maximum bandwidth variance.

10.2 FROM THE STACK-ROOFLINE MAX TO THE ADDITIVE POLYNOMIAL (EQ. 1 → EQ. 2)

This subsection makes explicit the derivation connecting the stack-roofline max decomposition and the additive polynomial fit, and states the conditions under which the quadratic term matters and when linear terms suffice.

Phase decomposition. A request is a prefill phase of P tokens followed by a decode phase of G generated tokens (in this paper's protocol, $G = 64$ throughout; the decode-length invariance of Section 6.9 shows the structure is stable at $G \in \{128, 256, 512\}$ as well). For each step, the stack-roofline model (Eq. 1) gives $T_{\text{step}} = \max(T_{\text{compute}}, T^{\text{visible}}_{\text{transport}}, T_{\text{kv-miss}}, T_{\text{sync}})$; by design $T_{\text{kv-miss}} = 0$ on hot-tier hits and T_{sync} is amortized below the noise band (measured: plan-time overhead $4\,\mu$s/request), so the binding pair is compute vs. visible transport.

Prefill. At prefill step i (the i-th token being encoded), the attention work and the relayed traffic both grow with i: compute $\propto i$ (attention over the growing prefix) and, for the staged-split, relayed KV bytes $\propto i$. Summed over the P-token prefill, each contributes $\propto P^2$: the quadratic term. With compute/transport overlap, the visible part of the max is only the non-hidden fraction; the fit absorbs the hidden fraction into the constant.

Decode. Each of the G decode steps attends over the full ctx prefix. In the staged split the boundary activation is handed to the next stage once per generated token—a constant-size payload—and each stage reads the KV shards *it owns* from its own memory; the implementation never relays remote KV (Appendix 24). The visible per-token cost is therefore the handoff plus its synchronization, constant in ctx: this is the per-token transport tax τ, and it is what the fitted slope difference $b_S - b_R$ measures. The decode attention read does grow with ctx (memory bandwidth for the KV read), but both modes pay it, so it cancels in the difference $D(\text{ctx})$ and does not enter τ; the curvature gap $\Delta c > 0$ comes from the prefill phase, where the per-layer hidden-state handoffs are paid once per prefilled token and therefore sum to $\propto P^2$ (the $O(P^2)$ label of Eq. (S27)).

Sum and conditions. Writing the request latency as the sum of the two phases plus a fixed per-request overhead (plan, dispatch, model warm, tokenizer) reproduces the fitted form (Eq. 2):

$$T(\text{ctx}) = A + \underbrace{(G\,\tau_{\text{step}} + \beta)\,\text{ctx}}_{b\,\text{ctx}} + \underbrace{c\,\text{ctx}^2}_{O(P^2)\ \text{prefill}}, \tag{S27}$$

where β collects the per-token non-transport prefill/decode cost and c the attention-quadratic residual (the split's extra attention traffic after overlap). Three conditions make the fit valid: (i) one term dominates per phase (no fine-grained interleaving that would break the sum of maxes into a max of sums); (ii) the decode length G is fixed across the calibration ladder—it is, by protocol, and the decode-length matrix shows the per-token structure is stable; (iii) prefill/decode disaggregation: prefill contributes the quadratic, decode the linear, and the fitted coefficients are the sum over phases, so a deployment that changes the prefill/decode mix re-scales b and c rather than invalidating the form (the concurrency and batch-penalty measurements are exactly such coefficient scalings, Section 6.9).

When linear suffices and when the quadratic matters. The quadratic term matters when prefill is compute/attention-bound and not hidden (long contexts on fast links); the linear term suffices when the visible relay dominates (narrow links, transport-bound decode), which is the regime of this paper's platform: $c_S\,\text{ctx}^2$ overtakes $b_S\,\text{ctx}$ only above $\sim$33K for the split, and the crossover law's root is set by the linear gap τ; the measured $c_S - c_R = 1.9{\times}10^{-5}$ shifts the root by $< 8\%$ (740 vs. 769 tokens). The LOCO appendix quantifies the price: over the measured 12–14 point ladders the quadratic identification is stable (Δc cross-fold CV $< 15\%$), while the intercept ΔA trades off against curvature, which is why the paper rests decisions on the measured grid rather than parameter identifiability alone (Appendix 5.2).

Identification of the transport tax. The protocol isolates per-token transport from the setup premium as follows. (i) Per-mode ladders at 14 contexts, generation-forced 64-token completions, $n{\geq}5$ warm reps; six-parameter least squares separates the fixed premium A_m (setup, tokenizer, dispatch) from the per-token slope b_m and curvature c_m. (ii) The transport gap $\tau = b_S - b_R$ is identified against the replica, whose physical mode has no cross-device relay; the decode-length matrix (Section 6.9) confirms the linear-in-ctx attribution is not an artifact of a single decode length. (iii) The held-fit ($\tau_{\text{held}} = \text{mean}_{\text{ctx}}[(D + \Delta A - \Delta c\,\text{ctx}^2)/\text{ctx}]$, $\Delta A, \Delta c$ fixed at calibration) isolates the bandwidth axis: across the Gen4/Gen3 BIOS pair the held τ moves $0.56 \to 1.04$ ms/tok (measured bandwidth $13.3 \to 6.65$ GB/s, ratio 2.00 vs. predicted 1.85), which the bandwidth probe (deterministic, no drift) confirms is a link-speed effect. With $G = 64$, the reported τ is the per-prompt-token gap under this protocol; the crossover law $\text{ctx}^* = \Delta A/\tau$ is protocol-consistent because both ΔA and τ are measured under the same G.

10.3 Five-tier state taxonomy

The runtime organizes every state object (KV pages, weights, activation scratch) into five tiers, ordered by access cost on the topology graph G (main paper, Section 4): **hot** state resides on the GPU that will consume it next and is directly addressable by the executing stage; **tail** state is GPU-resident but belongs to a non-running stage (e.g., KV of a paused replica or a completed prefix awaiting a reuse decision) and is the first eviction candidate under memory pressure; **compressed** state is a lossy, reduced-precision or summarized representation whose restore cost is lower than recompute but whose use is a quality–cost trade-off, so it is only chosen when the mode score prices it in; **host RAM** is the warm tier—exact prefix KV evicted from GPU under LRU pressure lands here, and **Reuse** restores it with a single copy-engine transfer that is scheduled to overlap compute; **disk** state is the persistent tier (checkpointed weights, serialized prefix stores) with the highest restore latency, used for cross-process reuse and crash recovery rather than steady-state serving. **Place** assigns each object an initial tier under per-device budgets; **Move** promotes/demotes objects between tiers ahead of their consumer; **Recompute** treats rebuilding any non-hot object as a costed alternative to fetching it from a lower tier.

10.4 Bandit selector scores

The UCB selector (main paper, Section 4, Method) maintains a running average of the observed score for each mode plus an exploration bonus that decreases as the mode is selected more often. At

time step t the UCB score for mode m is

$$\mathrm{UCB}_m(t) = \bar{S}_m(t) + \beta\sqrt{\frac{\log t}{N_m(t)}}, \tag{S28}$$

where $\bar{S}_m(t)$ is the empirical average score, $N_m(t)$ the number of selections up to t, and $\beta > 0$ the exploration parameter; the selector chooses $m_t = \arg\max_m \mathrm{UCB}_m(t)$. In the Thompson Sampling implementation, the posterior per mode is Gaussian with mean equal to the empirical average and variance proportional to the inverse of the observation count; the selector samples one score per mode and chooses the argmax sample.

10.5 Per-request dispatch loop and the refit selector (pseudocode)

The dispatch loop (Algorithm S1) is the fixed per-request sequence of the main paper's Section 4; the selector (Algorithm S2) is the repaired, latency-structured version whose scores come from Eq. (2) of the main paper.

Algorithm S1 Per-request dispatch loop (main paper, Section 4)

Require: Topology graph G, workload w, selected mode m, plan P

Restore cached prefix KV from the warm tier if a prefix hit exists (exact rendered-prompt match)
Prefill through the stages, recording per-relay visible transport
if $m =$ staged **then**
    Decode with on-demand cross-device relay: layer activations hand off at the partition boundary; remote KV is relayed per token
else if $m =$ replica **then**
    Decode locally on the assigned worker (round-robin dispatch over replica workers)
else
    Decode locally on the single device (quantized model, full KV local)
end if
Cache the prefix KV; under memory pressure, evict LRU pages to the warm tier
return Generated tokens and telemetry (latency, per-GPU peaks, visible transport)

Algorithm S2 Greedy topology-aware mode selection with the refit selector (main paper, Section 4)

Require: Topology graph G, workload w, calibrated per-mode latency model, EWMA state, measured regime law

Compute per-mode hard feasibility masks (replica KV slope and ceiling; staged activation term and concurrency limits; 5% reserve)
for all eligible modes $m \in \{\text{single, staged, replica}\}$ **do**
    Predict $T_m(\text{ctx}) = A_m + b_m\,\text{ctx} + c_m\,\text{ctx}^2$ {Eq. (2), main paper}
    if inside the estimator's noise band ($|T_{\text{split}} - T_{\text{replica}}| \le 2\epsilon_t \approx 2{,}766$ ms) **then**
        $m_{\text{band}} \leftarrow$ mode from the measured regime law (replica at conc-1, split at conc-≥ 2, single when fastest)
        break {the band rule decides this request; the fit is not used}
    end if
    if $m =$ replica and concurrency ≥ 2 **then**
        Apply the measured batch-wall penalty
    end if
end for
$m^* \leftarrow m_{\text{band}}$ if the band rule fired, else $\arg\min_m T_m(\text{ctx})$ among budget-fitting modes
Execute m^*; observe the outcome and update the cost model (EWMA)
return m^*

11 PROOF OF THE CUMULATIVE PREDICTION-ERROR REGRET BOUND

We prove the prediction-error bound of the main paper (Section 5, result (ii)): under the uniform prediction-error assumption ($|\hat{V}_m^{(t)} - V_m^{(t)}| \le \epsilon_t$ for all modes m and steps t), the greedy selector $\hat{m}_t = \arg\min_m \hat{V}_m^{(t)}$ satisfies $|V_{\hat{m}_t}^{(t)} - V_{m_t^*}^{(t)}| \le 2\epsilon_t$ per step and hence $\sum_{t=1}^{T}(V_{\hat{m}_t}^{(t)} - V_{m_t^*}^{(t)}) \le 2\sum_t \epsilon_t$.

Proof. By definition of $\hat{m}_t$, we have $\hat{V}_{\hat{m}_t}^{(t)} \le \hat{V}_{m_t^*}^{(t)}$. Using the error bound twice:

$$V_{\hat{m}_t}^{(t)} \le \hat{V}_{\hat{m}_t}^{(t)} + \epsilon_t \le \hat{V}_{m_t^*}^{(t)} + \epsilon_t \le V_{m_t^*}^{(t)} + 2\epsilon_t, \tag{S29}$$

$$V_{m_t^*}^{(t)} \le \hat{V}_{m_t^*}^{(t)} + \epsilon_t \le \hat{V}_{\hat{m}_t}^{(t)} + \epsilon_t \le V_{\hat{m}_t}^{(t)} + 2\epsilon_t. \tag{S30}$$

Therefore $|V_{\hat{m}_t}^{(t)} - V_{m_t^*}^{(t)}| \le 2\epsilon_t$; summing over t gives the cumulative bound. The two chains are the two *directions* of the absolute-value bound, not competing orderings: the first uses $\hat{V}_{\hat{m}_t} \le \hat{V}_{m_t^*}$ from the argmin definition of $\hat{m}_t$, and the second uses that m_t^* is the true optimum, so $V_{m_t^*} \le V_{\hat{m}_t}$. Together they sandwich the gap; the first chain alone already bounds it, and the second is included only to make the symmetry explicit. □

12 SCALING LAW: DERIVATION, PROOF, SCOPE, AND PARAMETRIC SENSITIVITY

12.1 DERIVATION AND PROOF

Consider the decode stage of autoregressive inference. Let T_{comp} be the single-device compute time per token (including attention, MLP, and projection), and let W_{state} be the amount of cross-device state that must be transferred per token (e.g., hidden-state handoffs between pipeline stages). Let $B_{\min} = \min_{e \in E} B_e$ be the minimum edge bandwidth in the topology and $L_{\max} = \max_{P_{u,v}} L_{P_{u,v}}$ the maximum path latency.

Under k-device pipeline parallelism, the per-token compute time is T_{comp}/k (assuming perfect load balancing), and the per-token communication time is $W_{\text{state}}/B_{\min} + L_{\max}$ (assuming the critical path is the slowest edge). The optimal per-token latency is therefore the maximum of these two terms:

$$T_{\text{opt}}(k;\tau) = \max\left(\frac{T_{\text{comp}}}{k}, \frac{W_{\text{state}}}{B_{\min}} + L_{\max}\right). \tag{S31}$$

Equation (S31) reveals two regimes:

- **Linear regime** ($k < k_{\text{crit}}$): When $T_{\text{comp}}/k > W_{\text{state}}/B_{\min} + L_{\max}$, the compute time dominates and communication is effectively hidden. In this regime, throughput scales approximately linearly with k.
- **Saturation regime** ($k \ge k_{\text{crit}}$): When $T_{\text{comp}}/k \le W_{\text{state}}/B_{\min} + L_{\max}$, the communication time dominates and throughput saturates. Additional devices beyond k_{crit} provide diminishing returns.

The critical device count is

$$k_{\text{crit}} = \frac{T_{\text{comp}}}{W_{\text{state}}/B_{\min} + L_{\max}}. \tag{S32}$$

Proof of the scaling law (main paper, Section 5, result (iii)). From the stack-roofline model, the optimal per-token latency is the maximum of compute time and visible communication time. Under perfect load balancing, compute time is T_{comp}/k. The visible communication time is at least $W_{\text{state}}/B_{\min} + L_{\max}$ because the slowest path in the topology bounds the transfer time. Taking the maximum gives Eq. (S31). The maximum is below T_{comp} if and only if both terms are: the compute term requires $k > 1$, and the communication term rearranges to the stated effectiveness budget. Complete hiding requires the compute term to dominate, which is the regime boundary $k < k_{\text{crit}}$ of Eq. (S32) and rearranges to the stronger budget. □

12.2 SCOPE AND THE MODE CROSSOVER

The scaling law compares k-device stacking against single-device execution of the *same* total state and assumes (i) perfect compute/transport overlap and (ii) that the workload mandates stacking. It therefore does not model the request-parallel replica. We close that gap with a two-mode law over the per-request latency structure the refit selector uses (main paper, Section 6.5). Write the per-request latency of the k-device split and the single-card replica as

$$T_{\text{split}}(\text{ctx}) = A_S + b_S\,\text{ctx} + c_S\,\text{ctx}^2, \qquad T_{\text{replica}}(\text{ctx}) = A_R + b_R\,\text{ctx} + c_R\,\text{ctx}^2, \tag{S33}$$

with measured coefficients on the representative platform (PCIe 4.0 x8 relay, Qwen2.5-7B 4-bit): $A_S = 2{,}331.5$ ms, $A_R = 2{,}615.3$ ms, $b_S = 0.9290$ ms/token, $b_R = 0.5597$ ms/token, $c_S = 2.752{\times}10^{-5}$, $c_R = 8.339{\times}10^{-6}$ ms/token^2. Define the per-token transport tax $\tau = b_S - b_R = 0.369$ ms/token, the fixed-cost gap $\Delta A = A_R - A_S = 283.8$ ms, and the transport-growth gap $\Delta c = c_S - c_R > 0$.

Theorem 12.1 (Mode crossover)**.** *If $\tau > 0$, $\Delta c > 0$ and $\Delta A > 0$, the latency difference $D(ctx) = T_{split} - T_{replica}$ has exactly one positive root ctx*—the solution of $\Delta c\,ctx^2 + \tau\,ctx - \Delta A = 0$—with the split preferred for ctx $<$ ctx* and the replica for ctx $>$ ctx*, while both modes are feasible. For small $\Delta c\,ctx^2$ the root reduces to the linear law ctx$^* \approx \Delta A/\tau$. The premise $\Delta A > 0$ is not implied by the other two and is necessary: if $\Delta A \le 0$ then $D(0) = -\Delta A \ge 0$ while $D'(ctx) = \tau + 2\Delta c\,ctx > 0$, so D has no positive root and the replica is never worse—the fast-link regime measured on the 25.4 GB/s host, where $\Delta A < 0$ (we report that case as "no root"). With $\Delta A = 0$ the root degenerates to ctx$^* = 0$.*

Proof. $D(0) = -\Delta A < 0$, $D'(\text{ctx}) = \tau + 2\Delta c\,\text{ctx} > 0$ for ctx > 0, and $D \to \infty$; hence exactly one positive root with the stated sign pattern. □

The linear law makes the topology dependence explicit: the transport tax scales inversely with the effective bandwidth, $\tau(B) = \tau_0 \cdot (B_0/B)$, while ΔA is bandwidth-independent, so **the crossover grows linearly with bandwidth**: ctx$^*(B) = \Delta A\,B/(\tau_0 B_0)$. On the measured platform ctx$^* \approx 740$ tokens—below the smallest evaluated workload and inside the estimator's noise band (main paper, Section 6.5), which is why the replica wins every measured feasible cell.

Feasibility ceiling. The replica's capacity bound is memory-side and bandwidth-independent:

$$\text{ctx}^*(\text{rep}) = (M_{\text{usable}} - W_{\text{model}} - w_0)/s_{\text{rep}},$$

with w_0 the fixed working-memory term (activations, allocator and runtime overhead, calibrated once per engine) and s_{rep} the measured per-model replica state slope (191.7 KiB/token on Qwen $= 3.424\times$ the bf16 KV size; 250 KiB/token on Llama $= 1.91\times$; learned per model via EWMA at runtime). On the platform ctx*(rep) $\approx$ 25.3K tokens, inside the measured bracket (fits at 24,576; OOM at 26,000 and 28,000). Stacking's capacity grows with k, and the k_{crit} law bounds how many devices remain useful before saturation.

Three-regime map. Combining the two laws: (1) fixed-cost regime (ctx $<$ ctx*), where the split's smaller fixed cost can win; (2) replica regime (ctx$^* \le$ ctx $\le$ ctx*(rep)), zero per-token transport dominates (measured 14/14 on this platform, 512–24,576); (3) capacity regime (ctx $>$ ctx*(rep)), where the replica OOMs and stacking wins by feasibility.

Topology sweep (selector decisions, CPU-only planner sweep; $\tau(B) = \tau_0 B_0/B$, ΔA fixed, ceiling 25,257 tokens):

The sweep formalizes the paper's thesis: on narrow interconnects the fixed-cost regime collapses to a few hundred tokens and the replica dominates whatever it can fit; on NVLink-class links the split wins through multi-thousand-token contexts and the classical U-shape re-emerges. The datacenter intuition that "splitting wins" is a high-bandwidth artifact: the topology, through ctx$^*(B)$ and ctx*(rep), decides the mode. All numbers reproduce via `scripts/verify_mode_crossover_law.py` (`reports/selector_refit/mode_crossover.json`).

Concurrency (measured law). At concurrency ≥ 2 the replica's zero-transport advantage is overridden structurally: every worker pays the full prefill on one card with no transport hiding (measured

Table S26: Mode-crossover sweep: selector decisions across synthetic topologies (4-bit, concurrency 1). rp = request-parallel replica. The crossover moves right with bandwidth; the ceiling is bandwidth-independent.

Topology	B (GB/s)	τ (ms/tok)	ctx* (linear)	ctx* (quadratic)	ceiling
USB2 hub	0.06	98.47	3	3	25,257
USB4/Thunderbolt	3	1.97	144	144	25,257
PCIe 3.0 x16	16	0.37	769	740	25,257
PCIe 4.0 x8 (calibrated)	16	0.37	769	740	25,257
PCIe 4.0 x16	32	0.18	1,537	1,348	25,257
PCIe 5.0 x16	64	0.09	3,074	2,131	25,257
NVLink	200	0.030	9,607	3,153	25,257

$1.84\times$ batch wall vs. the split at 1–2K contexts), while the split's batch efficiency improves with context (per-request factor $1.5\times$ at $\leq$2K $\rightarrow$ $0.64\times$ at $\geq$16K: fixed decode/relay costs amortize). Measured: the split wins all six stability cells (1,024/2,048/16,384 $\times$ concurrency 2/4) plus both fresh concurrency-8 cells (4,096/8,192). We state this as a measured regime law with the mechanism identified; a closed-form batch model is future work.

Relation to the time-scaling ("τ") theory. Our transport tax τ shares its name and spirit with the data-movement time constant of He's time-scaling theory (He, 2026), which diagnoses data movement—not compute—as the binding constraint of AI infrastructure (over 80% of cluster energy spent on data movement) and proposes compressing the system-level τ. Our measurements are best read as a single-link, decision-level instance of that constant. The connection is threefold, and all three parts are measured in this paper: (i) the diagnostic holds at the smallest scale we can measure—on a narrow PCIe link, the per-token relay tax, not the compute, decides whether the second GPU helps (Section 6.5 and the measured regime law); (ii) our crossover law quantifies the *decision consequence* of that constant: ctx$^* = \Delta A/\tau$ maps a link's τ onto a mode boundary, and a policy that halves the relayed bytes (the theory's compression agenda, in miniature) doubles ctx* (transport-tax reduction, main paper); and (iii) four measured bandwidth regimes—Gen3 x8 (6.65 GB/s) and Gen4 x8 (13.3 GB/s) on the workstation, a cross-NUMA Gen4 x16 on a second host (17.6 GB/s), and a no-NVLink $4\times$RTX 4090 host (25.4 GB/s, Appendix 20.2)—anchor the $\tau \propto 1/B$ scaling the theory assumes at system level, at single-machine level: the same-host held pair scales $1.85\times$ against the $2.0\times$ predicted, and at 17.6 GB/s the free fit yields $\tau \leq 0.187$ ms/token, a bound $3\times$ below the held 0.560 at 13.3 GB/s (direction consistent; the magnitude is not identifiable across hosts, as disclosed in the second-host paragraph). Where the theory compresses τ in hardware, this paper's decision layer takes the topology's τ as given and makes the optimal mode decision under it—the two agendas compose.

Reasoning-style workloads amplify the tax. The per-token tax is linear in decode length, so workloads that emit long chain-of-thought blocks (reasoning-style models) multiply the decode-token count by an order of magnitude and pay the transport tax that many more times; for such workloads the split-vs-replica balance may shift for reasons orthogonal to bandwidth. This is unmeasured here—every experiment in this paper uses the short, 64-new-token generation protocol by design—and is an open direction the law's structure predicts but does not yet verify.

12.3 Parametric sensitivity across seven topologies

k_{crit} depends on four parameters (T_{comp}, W_{state}, $B_{\min}$, $L_{\max}$) that vary across topologies and workloads; rather than claiming a single measured k_{crit} on one platform, we compute it across physically grounded ranges. Parameter ranges: T_{comp} 50–500 ms/token (single-device decode for 7B–70B models); s_{eff} 1 KB–1 MB/token (the state that actually crosses the slowest edge: the boundary handoff; large values model designs that do carry KV across the link); $B_{\min}$ 0.06–200 GB/s (USB2 to NVLink); $L_{\max}$ 0.01–1.0 ms (direct P2P to USB hub).

k_{crit} spans more than two orders of magnitude across topologies, and in every row it sits far above the device counts a consumer deployment can realise ($k \leq 8$): throughput is linear for $k \leq 8$ on USB4, PCIe and NVLink alike, and marginal only above $k \approx 150$ even on a 480 Mb/s USB2

Table S27: Parametric sensitivity of k_{crit} across representative topologies (fixed $T_{\text{comp}} = 167$ ms, $s_{\text{eff}} = 3.2$ KB/token boundary handoff; see the correction note for the earlier 31 MB version of this caption: it used the 512-token sequence footprint as a per-token cost, batch size 1). k_{crit} values are computed directly from Eq. (S32).

Topology	$B_{\min}$ (GB/s)	$L_{\max}$ (ms)	k_{crit}	**Regime for typical** k
USB2 hub (480 Mb/s)	0.06	1.0	159	linear to $k \approx 150$
USB4/Thunderbolt edge	3	0.5	333	linear for all $k \leq 8$
PCIe 3.0 x16 (legacy)	16	0.35	477	linear for all $k \leq 8$
PCIe 4.0 x8 (consumer)	16	0.35	477	linear for all $k \leq 8$
PCIe 4.0 x16 (cloud VM)	32	0.20	835	linear for all $k \leq 8$
PCIe 5.0 x16 (future)	64	0.15	1,113	linear for all $k \leq 8$
NVLink (datacenter)	200	0.01	16,684	latency-floored, linear throughout

link. The dominant term is the path latency $L_{\max}$, not the state size: for a pipeline split the per-token payload that crosses the slowest edge is the boundary handoff, $s_{\text{eff}} \approx 3.2$ KB/token, whose bandwidth term is $s_{\text{eff}}/B_{\min} = 0.0002$ ms/token at 16 GB/s—exactly the number the transport-telemetry appendix measures—four orders of magnitude below $L_{\max} = 0.35$ ms. The per-card KV footprint (57,344 B/token, 61,184 with metadata) is *not* on this path in our implementation (KV stays on the layer that owns it), so it does not enter s_{eff}; an earlier draft of this appendix used the 512-token sequence footprint (31 MB) as if it were a per-token cost, mixing the two, and the numbers here are corrected accordingly. On the measured platform the *effective* per-token cost is the transport tax of the main paper, $\tau \approx 0.37$ ms/token, which is overhead- and latency-dominated rather than byte-dominated; substituting it for that whole term gives $k_{\text{crit}} \approx 167/0.37 \approx 451$ for the consumer topology, again two orders of magnitude above $k \leq 8$. Both readings agree: narrow-link stacking saturates because of per-handoff overhead and path latency, not because the link cannot carry the bytes.

What binds. With the corrected payload, $k_{\text{crit}} = T_{\text{comp}}/(s_{\text{eff}}/B_{\min} + L_{\max})$ is bounded above by $T_{\text{comp}}/L_{\max}$ and gains from bandwidth only while $s_{\text{eff}}/B_{\min}$ is comparable to $L_{\max}$—true only below $B_{\min} \approx 1$ GB/s for a handoff-sized payload. For a fixed topology, increasing batch size increases the per-token payload and therefore reduces k_{crit} *inversely*, $k_{\text{crit}} \propto 1/(c + s/B_{\min})$ with $c = L_{\max}$, not linearly. For a fixed workload, upgrading PCIe 3.0 to PCIe 4.0 doubles the bandwidth term, but that term is under 0.1% of c here, so k_{crit} moves by less than a percent—which is itself the finding: on narrow links the payoff of additional devices saturates because of latency and per-handoff overhead, not because the link cannot carry the state.

12.4 Measured compute and transport on the representative platform

On the representative platform (PCIe 4.0 x8, $B_{\min} = 16$ GB/s, $L_{\max} = 0.35$ ms), the measured per-token compute time—end-to-end latency amortized over generated tokens at the largest verified bf16 context—is $T_{\text{comp}} \approx 165$ ms/token (10,578 ms for 64 new tokens at 3,083 true tokens; Table S43 in this supplement). The nominal $T_{\text{comp}} = 167$ ms used in the sensitivity table differs by 1%, shifting k_{crit} by the same 1% (e.g., $477 \rightarrow 472$ for this topology). The telemetry session underlying the transport figure (single runs, no repetitions, so no standard deviations) reports, for the 128-token call, `total_ms` $= 8{,}347.9$, `ttft_ms` $= 1{,}030.9$, and 15.33 tok/s; in the 512-token call, `transport_visible_ms` $= 196.5$: 0.6% of the 31.9 s total, i.e., ≈ 3.1 ms of visible transport per generated token ($196.5/64$). This session predates the verified distinct-prompt matrices and mixed protocols, so we use it only as an upper-bound illustration of transport amortization: visible transport is a small fixed overhead whose share of total latency shrinks as context length increases, consistent with the verified matrices in which prefill compute dominates.

13 Cross-model validation: full table and per-cell numbers

Corrected model parameters. The per-token KV cache size is layers $\times$ 2 (K+V) $\times$ KV heads $\times$ head dim $\times$ 2 bytes. For Qwen2.5-7B (28 layers, GQA with *4* KV heads) this is $28 \times 2 \times 4 \times 128 \times 2 = 57{,}344$ bytes; for Llama-3-8B (32 layers, GQA with 8 KV heads) it is $32 \times 2 \times 8 \times 128 \times 2 =$

Table S28: Cross-model comparison on the representative topology (2×RTX 3060, PCIe 4.0 x8). Architecture rows follow the KV formula in the text; the weight footprint is parameters × 2 bytes (bf16). Result rows are *measured* under the verified true-token protocol (max_new_tokens = 64, distinct prompts). †Calibrated plan-time feasibility-gate estimate of the maximum admitted context, not a measurement.

Property	**Qwen2.5-7B**	**Llama-3-8B**
Parameters	7B	8B
Layers	28	32
Attention	GQA (4 KV heads)	GQA (8 KV heads)
Native context (tokens)	32,768	8,192
Vocab size	152k	128k
Weight footprint, bf16 (MiB)	14,534	15,316
Per-token KV (bytes)	57,344	131,072
Feasible context, 4-bit staged (measured)	65,536	65,536
Binding constraint	harness wall-clock limit at 65K (no OOM observed)	
Gate-estimated max context $\ell_{\max}$†	249,214	107,729
HF `device_map="auto"` 4-bit	OOM at 8,192	OOM at 8,192
bf16 warm latency @3,072 tok (ms)	$10{,}578 \pm 2{,}642$	$7{,}184 \pm 59$
4-bit latency @49,152 tok, process-cold (s)	150.9–152.7	151.3–151.5
4-bit latency @65,536 tok (s)	180.0 ± 0.5	226.9 ± 0.6
4-bit peak @65,536 tok (MiB/GPU)	4,419 / 6,652	6,960 / 9,095

131,072 bytes. Llama-3-8B therefore carries 2.29× *more* KV state per token than Qwen2.5-7B. (An earlier version of this analysis used a 16×-too-small Llama KV figure that inverted the comparison; the numbers above are the corrected ones.)

Per-cell findings. First, the framework's central advantage reproduces across architectures: both models run every tested context through 65,536 true tokens in 4-bit staged mode, while the HF `device_map="auto"` baseline OOMs at 8,192 for *both* models. Second, Llama-3-8B's 2.29× KV penalty manifests in the *memory ceiling*, not in per-call latency: in bf16 staged mode Llama is 12–32% *faster* than Qwen at 512–3,072 tokens (4,614/4,837/7,184 ms vs. 5,251/5,781/10,578 ms), at 32,768 tokens in 4-bit the two are at parity warm-vs-warm (67.2 vs. 64.7 s), and at 49,152 both models were measured process-cold and are likewise at parity (150.9–152.7 vs. 151.3–151.5 s; Qwen's single warm repetition at that length, 117.2 s, has no Llama counterpart), and even at 65,536 tokens Llama is only 1.26× slower (226.9 vs. 180.0 s) despite carrying 2.29× the KV state per token. Prefill compute, not KV traffic, dominates latency at these scales; consequently the earlier prediction of a latency crossover $C_{\text{cross}} \approx 3{,}072$ is *not supported*—no crossover was observed. Third, the earlier 147K-token prediction for Llama, derived from the 16×-too-small KV figure, remains falsified: the calibrated plan-time gate admits at most 107,729 tokens for Llama-3-8B on this platform, 27% short of 147K, and measurement has reached 65,536 tokens with no OOM on either model.

13.1 Feasibility-gate numbers behind the main-text percentages

Against the measured binding-GPU peaks at 65,536 tokens, the calibrated gate estimates 6,569 MiB (Qwen; measured 6,652, −1.3%) and 8,954 MiB (Llama; measured 9,095, −1.6%). It correctly rejects genuinely infeasible plans: bf16 7B on a single GPU (16,765 MiB), an 8-bit 27B model whose weight footprint alone exceeds a single GPU's capacity, and Llama at 131,072 tokens (13,050 MiB). Gate-estimated maximum contexts: 249,214 (Qwen) and 107,729 (Llama) tokens. The gate's feasibility test requires estimated peak at most total memory minus a 1.5% metadata reserve and 512 MiB, with 1,792 MiB activation and 512 MiB allocator reserves.

13.2 Gate false-positive / false-negative audit across the full corpus

Motivation. How is the 95% budget margin set, and what are the false-positive/false-negative rates of the gate across models and sessions?

We audit every measured decision cell in the corpus (375 cells: the 24-cell main grid, three randomized off-grid sets, the Gen3/Gen4 fresh sessions, the cloud 7B/14B hosts, the local 14B leg, and

the Llama cross-model cells) against the gate's per-cell choice (scripts/`analyze_gate_fpfn.py`; reports/diagnostics/gate_fpfn_analysis.json). A *false positive* is a cell where the gate chose a mode (predicted it feasible) and the forced run OOM'd on every rep; a *false negative* is a cell the gate rejected that a forced run completed.

Rates. 32 FPs and 1 FN across 375 cells. The FPs concentrate where the corpus goes *beyond* the concurrency band the paper's grid tests: 31 of 32 sit in exploratory off-grid cells at concurrency ≥ 3 and context $\geq 14{,}336$ (the request-parallel replica at c3/c6 and ctx $\geq$ 20K; the staged-split at 18,432×c3– the off-grid miss recorded in this supplement–plus 6 *marginal* cells at 10,240×c6 whose runs succeed 2–3 of 5 reps), and the 32nd is the local-14B staged cell at 49,152×c1. Inside the tested c1/c2/c4 band the gate is clean: 0 FPs on the 24-cell main grid (the 16,384×c4 both-OOM cell is *correctly rejected*), and the only c1/c2/c4 FP in the entire corpus is the 14B cell above. The single FN is the disclosed Llama-65K override (Section 6.7): the gate is deliberately conservative for KV-heavy long contexts and the override is documented, not silently bypassed. A scoring nuance is reconciled here: 28 of the c3/c6 both-infeasible off-grid cells score *correct* as mode-choice cells (the predicted mode was not refuted by measurement); under a feasibility-prediction reading they are gate FPs, and we report both views.

Margin. Across the 289 feasible cells with recorded peaks, the measured binding-GPU peak runs at a median 46% of the 95% usable budget (min 17%, max 100.3%—the marginal 10K×c6 cell). The 95% usable-budget margin (1.5% metadata reserve + 512 MiB + 1,792 MiB activation + 512 MiB allocator reserves, above) is therefore validated at single-request concurrency, and the audit identifies the missing term: *per-concurrency activation*. The FPs all occur where the batch raises the activation footprint beyond the single-request allowance; the fix is to scale the activation reserve with the number of concurrent requests (the saturating activation term of Section 6.5 is the measured form), which we record as the gate's next calibration point rather than a silent repair of the audit.

14 Per-tier memory residency accounting at 65,536 tokens

Main paper, Section 6.2 (memory accounting, in prose). This appendix records the full arithmetic behind the 65K state budget and reconciles the measured binding-GPU peaks with the 16 GB host DRAM of the platform.

KV bytes per token (derived from model configs). KV/token = layers $\times$ 2 (K+V) $\times$ KV heads $\times$ head dim $\times$ 2 bytes (bf16). Qwen2.5-7B-Instruct (28 layers, GQA with 4 KV heads, head dim 128): $28 \times 2 \times 4 \times 128 \times 2 = 57{,}344$ B/token. Llama-3-8B-Instruct (32 layers, GQA with 8 KV heads): 131,072 B/token. At 65,536 tokens the full bf16 KV is $65{,}536 \times 57{,}344 = 3{,}758{,}096{,}384$ B $=$ 3.50 GiB (Qwen) and $65{,}536 \times 131{,}072 = 8{,}589{,}934{,}592$ B $=$ 8.00 GiB (Llama). A multi-head-attention counterfactual (32 KV heads, 28 layers) would be 458,752 B/token, i.e., 28.0 GiB at 65,536 tokens—the figure behind the common expectation that 65K bf16 KV cannot fit this machine; both served models use grouped-query attention.

On-device residency (measured). The 65K runs execute in 4-bit staged mode: each card holds a $\sim$2 GiB weight share plus its KV share and activation/allocator overhead. Measured per-GPU peaks (`torch.cuda.max_memory_allocated`, n=3 distinct-prompt reps, `reports/beyondcap/`): Qwen 4,419/6,652 MiB and Llama 6,960/9,095 MiB—both inside the 12-GiB envelope with 5.3 GiB and 2.9 GiB of binding-GPU headroom. The calibrated admission gate estimates 6,569 and 8,954 MiB for the binding GPU (-1.3%/-1.6% error).

Host DRAM (16 GB). Host memory is not on the live-KV path at these contexts. It carries the engine and OS footprint plus the *warm tier*: exact prefix KV pages evicted from GPU under LRU pressure, held under a configured cap (a fraction of GPU state budgets; the optional CPU-offload expansion is capped at 4 GiB in the current prototype, `src/tworope/runtime/memory.py`). Persistent prefix stores are disk-backed. In the 65K matrix every repetition uses a distinct prompt, so the warm tier is never exercised there; the two independent consistency checks are therefore the derived KV sizes above and the measured GPU peaks.

Table S29: Poisson-arrival simulation (medians over 20 seeds): TTFT P95 and SLO-violation rate per policy and arrival rate. Goodput tracks λ for all policies (aggregate capacity is not the binding constraint).

Policy	λ=0.3		λ=0.4		λ=0.5	
	P95 (s)	viol.	P95 (s)	viol.	P95 (s)	viol.
always-replica	10.1	1.5%	10.3	5.8%	12.8	16.3%
always-split	9.6	8.2%	15.7	27.8%	38.8	67.0%
threshold	**6.6**	**0%**	**7.4**	**0.7%**	**9.5**	**2.2%**
planner (load)	9.7	5.3%	15.3	22.9%	36.5	62.2%

15 Poisson-arrival simulation over the measured distributions

A discrete-event simulation extrapolates the measured per-request latency distributions (the llama.cpp tail study's 12 recorded requests per config and context) to a Poisson arrival process. Requests draw contexts $\{8{,}192\ (70\%), 16{,}384\ (30\%)\}$ and arrive at rate $\lambda \in \{0.1, \ldots, 0.5\}$ req/s over a 1,200 s horizon (100 s warmup, 20 seeds). Serving geometry matches the study: the replica pool has two single-GPU servers and the split pool one dual-GPU server, each with four slots; an 8K request occupies one slot and a 16K request two (reproducing the study's `-np 4` (8K) / `-np 2` (16K) configurations). Policies: always-replica, always-split, a context-threshold rule (8K→replica, 16K→split), and the planner's load-based emission (split when system-wide busy slots ≥ 2, the selector's concurrency rule). SLO: TTFT ≤ 6 s (8K) / ≤ 12 s (16K). Driver: `scripts/sim_arrival_goodput.py`; raw results: `reports/diagnostics/arrival_sim.json`.

Three findings. First, the context-threshold rule dominates every policy at every rate (TTFT P95 6.6–9.5 s and $\leq$2.2% violations at λ=0.5, vs. 12.8 s/16.3% for always-replica and 38.8 s/67.0% for always-split): routing the long contexts to the staged-split and the short ones to the single-GPU replicas keeps both pools below saturation. Second, always-split collapses under sustained arrivals: its single four-slot machine saturates on the 16K jobs (two slots each), so queueing dominates the tail—the single-request advantage of the split does not transfer to a mixed sustained workload with this slot geometry. Third, the planner's load-based rule (split when $\geq$2 slots are *currently* busy) is dominated by the context rule at every rate: at $\lambda \geq 0.3$ the two-busy condition is met by most arrivals, so it sends most requests to the four-slot split server and inherits most of the always-split collapse (violations $5.3/22.9/62.2\%$ vs. $8.2/27.8/67.0\%$ for always-split), while at λ=0.1 it keeps almost everything on the replica pool. The simulation therefore *refines* the emission rule: under sustained arrivals the planner should route per request by *context* (short→replica, long→split) rather than by instantaneous load—a rule the framework's per-request dispatch loop already supports, and one we record as the planner's arrival-mode heuristic for future deployment validation. Caveats: service times are empirical samples of 12 requests per cell; arrivals are simulated (the hardware study used a deterministic stagger); and no server-switching or KV-transfer cost is modeled.

15.1 Real-trace arrivals: Azure code trace, 2024

The arrival study above uses a Poisson process; does the picture hold under a real arrival process? We replay the *shape* of a production trace through the same serving model. Data: the public Azure LLM inference trace 2024 (Stojkovic et al., 2025) (CC-BY), code workload, one-week release; the reachable prefix from this network is the first ∼16 h (May 10, 00:00–16:01 UTC; 1.11M requests), used here. Its own context distribution is *entirely short-context*: min/median/p90/max $= 1/1{,}937/6{,}392/7{,}692$ tokens (52.5% $\leq$ 2,048, 81.3% $\leq$ 4,096, 100% $\leq$ 8,192), so every request is an 8K-class request and no request falls in the 16K class; generations are short too (median 9, p90 46, max 5,000 tokens; 93.2% $\leq$ 64). The inter-arrival shape is bursty (mean 52 ms, median 18 ms, p95 208 ms; native rate ≈19 req/s — several times the two-GPU pool's capacity), so only the shape is transferable: the empirical inter-arrival distribution is unit-normalized and re-scaled to each λ (the study's load points), as documented in `scripts/sim_trace_goodput.py`.

The service model is now *per-request*: each arrival carries its trace (context, generation) pair, the context is bucketed to the nearest measured cell (2K/4K cells measured with the same protocol as the tail study; 8K/16K cells existing), one of the cell's 12 recorded (ttft, total) pairs is drawn, and

Table S30: Real-trace arrivals (Azure code trace, 16h prefix): medians over 20 seeds; TTFT P95 and 8K-SLO violation rate per policy and re-scaled arrival rate. All trace requests are 8K-class, so the context-threshold rule coincides with always-replica (it has no long-context request to route); every request completes (goodput follows λ). The "calibrated" row is the decision layer's own criterion (lowest predicted per-request latency, Section 4) evaluated against the measured per-cell service times and the pool's current slot occupancy, in place of the occupancy test the planner row uses.

Policy	λ=0.2		λ=0.5		λ=1.0		λ=2.0	
	P95 (s)	viol.	P95 (s)	viol.	P95 (s)	viol.	P95 (s)	viol.
always-replica	4.15	0%	4.42	0%	4.42	0%	4.52	0.3%
always-split	2.83	0%	2.86	0%	2.93	0.1%	9.25	17.8%
threshold (= replica)	4.15	0%	4.42	0%	4.42	0%	4.52	0.3%
planner (load)	4.15	0%	4.14	0%	3.93	0%	8.91	17.0%
calibrated	**2.83**	0%	**2.82**	0%	**2.83**	0%	**2.86**	0%

the decode portion is scaled by the request's generation: $T = \mathrm{TTFT} + (T_{\mathrm{rec}} - \mathrm{TTFT}_{\mathrm{rec}}) \cdot G/64$. The new 2K/4K cells, measured under the same `-np 4` staggered protocol as the 8K/16K cells, show the split ahead even at short contexts (TTFT P50 $881/1{,}556$ ms vs. $1{,}048/2{,}180$ ms for the replica at 2K/4K)—the same batch-pressure shift the concurrency rule encodes; at concurrency 1 the single-request ladder places this engine's crossover in $\leq 1{,}024$.

Three honest readings. (i) **The real workload is entirely short-context**: the largest trace context is 7,692 tokens, below the 8K class and four orders below the paper's 65K target. On it, the context-threshold rule degenerates to always-replica, so this trace validates the arrival *process* (burst shape) and pool-saturation behavior, not long-context mode selection—the mode law's long-context half is exercised only by the controlled matrices (the trace's largest context is $\times 8.5$ *below* the paper's 65K target). This is stated to prevent over-reading; a trace containing $>$ 8K contexts (e.g. the conversation workload, not downloadable here) would exercise the split branch. (ii) **At the study's low load points the split is the right mode even here**: with the per-request model it wins TTFT P50/P95 at every $\lambda \leq 1.0$ (P95 2.8–2.9 s vs. 4.1–4.4 s; violations $\leq$0.1% both), matching the per-request engine measurements. (iii) At λ=2.0 the mechanism flips to pool saturation, exactly as in the Poisson study: the split pool's *single* four-slot server saturates (P95 9.25 s, 17.8% violations) while the replica pool's two four-slot servers hold (P95 4.52 s, 0.3%); the load-aware planner inherits the collapse (17.0%) because its two-busy threshold is met at nearly every arrival, and at $\lambda \leq 1.0$ it forfeits the split's per-request advantage (P95 3.9–4.2 s vs. 2.8–2.9 s) because the pool is below two busy slots most of the time. Replacing that occupancy test with the decision layer's own criterion—the mode with the smaller projected completion time, where the projection is this request's measured per-cell service time for that mode plus the wait implied by that pool's current slot occupancy—removes the failure at every load point (8.91 s/17.0% becomes 2.86 s/0.0% at λ=2.0, and the row is best or tied-best at every λ). The corollary for the planner is the same refinement the Poisson study records, and the pool geometry—not the per-request winner—determines the tail under sustained real arrivals. Caveats: this is the reachable 16-h prefix of the one-week file; arrivals are shape-re-scaled, not the trace's own rate; per-request service times are the measured 12-sample cells (near-idle λ values therefore correspond to the under-load cell values of the same protocol); and no server-switching or KV-transfer cost is modeled.

15.2 Mixed-context arrivals: what a short-context trace cannot test

The trace above is entirely short-context, so its routing column is vacuous: with no request above the 8K class, the context rule and always-replica make the same decision on every arrival, and the study says nothing about *routing*. We therefore give the *same* arrival process a workload that does contain long contexts, using only cells already measured.

Construction. The inter-arrival process is unchanged (the trace's empirical distribution, unit-normalized and re-scaled to each λ). Each arrival draws its $(\mathrm{ctx}, \mathrm{gen})$ pair from a mixture: with probability $1-f$ the trace's own pair, and with probability f a *long tier* whose context is uniform on $[12{,}288, 16{,}384)$—the 16K class, the longest measured under this protocol—and whose generation is drawn from the trace's own distribution, so the two tiers differ in context and not in decode

Table S31: Mixed-context arrivals: TTFT P95 in seconds, with the 8K/16K SLO violation rate (%) in parentheses; medians over 20 seeds. Same arrival process as Table S30; f is the synthesized long-context share. The calibrated row is the decision layer's criterion (lowest predicted per-request latency, Section 4) evaluated against the measured per-cell service times and the pool's current occupancy.

Policy	f=0.1, λ=1	f=0.25, λ=0.5	f=0.25, λ=1	f=0.5, λ=0.5
always-replica	9.63 (3.1)	10.33 (5.2)	25.87 (51.3)	22.20 (35.1)
always-split	13.81 (22.5)	13.96 (19.6)	205 (100.0)	68.05 (86.5)
context-threshold	6.04 (0.1)	8.11 (1.5)	21.68 (12.4)	30.24 (27.4)
occupancy planner	12.90 (19.1)	13.22 (15.7)	202 (100.0)	66.75 (85.2)
calibrated (measured cost)	**6.40** (0.0)	**8.09** (0.0)	**9.38** (1.0)	**9.72** (1.7)

length. f is a controlled knob, not an observed quantity; the reported points are $f \in \{0.1, 0.25, 0.5\}$, and the f=0 endpoint is the pure-trace study of Table S30, reported there rather than duplicated here. Service times are the same measured cells through the same per-request model; nothing is extrapolated.

The routing column stops being degenerate. On the pure trace the context rule routes 0% of arrivals to the split pool—it *is* always-replica (Table S30). With a long-context share it becomes a real decision: Table S32 reports, per policy, the fraction routed to split, which is 10.1% at f=0.1, λ=1, 24.7–25.3% at f=0.25, and 50.4% at f=0.5, λ=0.5—the long tier, plus, for the load-aware policies, short-context splits.

Pool geometry, not the per-request winner, sets the tail. A 16K request takes two of a split server's four slots, so the split pool serves at most two concurrent 16K requests, while the replica pool serves eight (one slot each, across two four-slot servers). The split's per-request advantage therefore does not survive load: at f=0.25, λ=1.0, always-split runs to a 205 s P95 with 100% SLO violations while always-replica holds 25.9 s/51.3%, and it is the calibrated row of Table S31—not either fixed policy—that holds 9.4 s/1.0%.

The occupancy stand-in collapses; the calibrated rule does not. The load-aware planner keys on pool occupancy (*split once two slots are busy* pool-wide), which is satisfied at nearly every arrival once the pool is loaded; it therefore drifts towards always-split and inherits the collapse (202 s/100% at f=0.25, λ=1.0, and 99.4% of arrivals routed to the smaller pool). The calibrated rule—the decision layer's own criterion, i.e. the mode with the smaller projected completion time from this request's measured per-cell service time plus the wait implied by the pool's current occupancy—is best or tied-best on the violation rate at every point of the mixed sweep, and on the pure-trace study it is best at every λ (Table S30).

Honest limits. (i) The long share f is synthetic and swept, not observed: the code workload contains no long-context request at all and the conversation workload is not reachable from this network, so the trace's own long-context share is unknown. (ii) The long tier is the 16K class because that is the longest class measured under this protocol, so the 16K–65K range is still carried by the controlled long-context matrices rather than by this arrival study. (iii) Beyond $f \approx 0.25$ at λ=2.0 the two-GPU pool is over capacity on slot arithmetic alone and no routing rule recovers the tail. (iv) That last regime is also where the choice stops being a preference: above the measured replica ceiling (Appendix 4.4: a 24,589-token request serves at 10,014 MiB peak, a 32,781-token request OOMs) the split is not the better mode but the only one—the feasibility gate, not the crossover, is what routes there.

15.3 Load-aware crossover sensitivity: bandwidth × batch pressure × arrival rate

Motivation. How robust is the crossover ctx^* to workload variability in continuous batching (changing batch sizes, arrival rates)? A sensitivity surface of ctx^* vs. (B, batch size, arrival rate) on one engine makes this explicit.

We extend the arrival simulation above with two additional dimensions, anchored on the same measured llama.cpp distributions and the calibrated transport model (τ_0 =

Table S32: Fraction of arrivals each policy routes to the split pool (%) on the same four mixed-context settings. On the pure trace of Table S30 this column is empty: every request is 8K-class, so the context rule is indistinguishable from always-replica—the degeneracy this study exists to remove.

Policy	f=0.1, λ=1	f=0.25, λ=0.5	f=0.25, λ=1	f=0.5, λ=0.5
always-replica	0.0	0.0	0.0	0.0
always-split	100.0	100.0	100.0	100.0
context-threshold	10.1	24.7	25.3	50.4
occupancy planner	82.7	76.5	99.4	97.5
calibrated (measured cost)	59.4	65.7	42.4	56.4

Table S33: Sensitivity of the load-aware optimal threshold θ^* to bandwidth, batch pressure, and arrival rate (medians over 20 seeds; SLO violation rate in parentheses). The two-level context mix (70% 8K / 30% 16K) resolves the decision boundary coarsely; the single-request law's $\mathrm{ctx}^*(B)$ is shown for reference.

B (GB/s)	pressure	λ=0.2	λ=0.35	λ=0.5	ctx^*(1req)
6.65	1.0	θ^*=12–20K (0%)	12–20K (3.3%)	12–20K (16.3%)	384
6.65	1.5–2.0	SLO envelope exceeded (violation → 1.0 for every θ)			384
13.3	1.0	12K (0%)	12K (0.5%)	12K (2.2%)	769
13.3	1.5	6K (22.6%)	12K (78.8%)	12K (88.5%)	769
13.3	2.0	6K (61.1%)	12K (95.0%)	12K (99.8%)	769
26.0	1.0	6K (0%)	6K (0%)	12K (0%)	1,503
26.0	1.5	6K (0.9%)	6K (7.3%)	6K (42.1%)	1,503
26.0	2.0	6K (7.1%)	6K (41.5%)	12K (77.6%)	1,503

0.369 ms/tok at $B_0 = 13.3$ GB/s; scripts/`sim_sensitivity_surface.py`; reports/diagnostics/sensitivity_surface.json): (i) the *bandwidth* axis $B \in \{6.65, 13.3, 26\}$ GB/s (Gen3 x8 / Gen4 x8 / a nominal 2× point; the measured x16 relays are 17.6 and 25.4 GB/s, Appendix 20.2), which scales only the split's service times by the transport tax $\tau(B) = \tau_0 B_0/B$ (the replica relays nothing); and (ii) a *batch-pressure* factor $f \in \{1.0, 1.5, 2.0\}$ multiplying both modes' service times, standing in for the service-time inflation of denser batches (planner-level parameter). For each (B, f, λ) triple we grid-search the context threshold $\theta \in \{6{,}144, 8{,}192, 12{,}288, 16{,}384, 20{,}480\}$ of the "short→replica, long→split" rule to minimize the TTFT-SLO violation rate (20 seeds, 1,200 s).

Four honest readings. (i) **The load-aware threshold is stable across the measured bandwidths**: at f=1 the optimal θ^* sits on the 8K/16K boundary for both Gen3 and Gen4 (the two-level mix cannot resolve the single-request law's 384→769 movement), and drops below 8K only at 2× bandwidth—the same direction as $\mathrm{ctx}^*(B)$, confirming that the decision boundary follows bandwidth, not that the coarse mix is sensitive to it.

(ii) **The crossover metric matters**: the TTFT-SLO-optimal rule sends 16K to the split even where the completion-latency law says the replica wins (Gen3), because the split's prefill is shared across two GPUs and its TTFT stays under the 12 s budget—the load-aware crossover is *metric-dependent*, and the paper's latency-law crossover is the completion-time statement.

(iii) **Batch pressure is the sensitive direction**: at $f \geq 1.5$ the Gen3 envelope is exceeded wholesale (violation rate →1.0), and at Gen4/x16 θ^* collapses to the 6K boundary with violations rising to ∼78–99% at λ=0.5: the decision rule degrades gracefully in *direction* (more split at high bandwidth) but the SLO budget itself is load-limited.

(iv) The threshold rule remains the best policy wherever the envelope is not exceeded, consistent with Table S29. These are planner-level statements over the measured distributions; the batch-pressure axis is a parametric stand-in for measured concurrency effects (Section 6.9) and should be read as a sensitivity, not a measurement.

Table S34: Bootstrap convergence of the held-fit τ (relative to the full-calibration $\tau = 0.369$ ms/tok; 200 draws; decision-flip rate = fraction of bootstrap fits whose split/replica verdict differs from the full calibration on the 14-context grid). Startup cost is the estimated wall time of a two-mode probe ladder at those contexts.

probe contexts k	τ bias	τ std	decision-flip rate	est. startup
2	$+1.8\%$	14.9%	0.4%	$\sim$10 min
3	-0.9%	12.7%	0.4%	$\sim$46 min
4	-1.8%	12.1%	0.3%	$\sim$56 min
5	$+0.3\%$	10.8%	0.2%	$\sim$78 min
7	$+0.3\%$	8.4%	0.0%	$\sim$70 min
10	$+0.4\%$	7.0%	0.0%	$\sim$132 min

16 SELECTOR COLD-START AND REGIME-SHIFT RECOVERY (SIMULATION)

To answer "how many observations before decisions stabilize on a new engine/model" without new hardware, we simulate the latency-structured selector on the paper's calibrated ground truth ($T_{\text{split}} = 2{,}331.5 + 0.9290\,\text{ctx} + 2.752\times10^{-5}\text{ctx}^2$; $T_{\text{replica}} = 2{,}615.3 + 0.5597\,\text{ctx} + 8.339\times10^{-6}\text{ctx}^2$; observation noise $\sigma = 1{,}383$ ms), log-uniform contexts in $[512, 32{,}768]$, 1,000 rounds, 20 seeds (`scripts/sim_cold_start_ewma.py`).

Cold start. Each selector starts from a prior whose transport tax is biased by a factor $x \in \{0.4, 0.7, 1.0, 1.5\}$ and updates by least-squares on its observation buffer (full or sliding windows $W \in \{25, 50, 100\}$). The convergence horizon—first round after which the rolling-100 decision accuracy stays above 90% for 50 rounds—is **100–160 observations** (median over seeds): 129 at the most adverse prior ($0.4 \times \tau$, which inverts the largest decision region), 106–113 at unbiased or mild priors, and windows matter little for cold start (W=25 adds $\sim$30 rounds at the worst prior). Final rolling accuracy is ≥ 0.99 for every configuration.

Regime shifts. At round 500 the link "upgrades" ($\tau \to 0.5\tau$), flipping the true winner from replica to split for $\sim$73% of contexts. The execute-mode-only greedy selector is *blind* to this: it never samples the currently losing (split) arm, so its split estimate stays stale and rolling accuracy collapses to 0.37 with a persistent $\sim$215 ms excess latency per decision—independently of window size. Adding exploration (ε=0.1) or periodic dual-mode calibration every 20–50 rounds only partially recovers (accuracy 0.46–0.59), because single-context refresh points leave the slope estimate dominated by stale observations; a full ladder refresh (both modes at six contexts every 200 rounds, the paper's fresh-cell protocol) restores accuracy between passes with the expected sawtooth decay. The practical lesson matches the paper's design: decisions inside the noise band are made by *measured* regime laws, and periodic fresh-cell re-measurement—not exploration bonuses—is the recovery mechanism; automating its cadence is future work. The recommended production recipe these runs imply: W=100 sliding least squares with a prior seeded from the topology's $\tau_0(B)$ estimate (main paper, crossover law), expecting 100–160 observations to converge on a new engine/model and scheduling a full-ladder dual-mode refresh every 200 rounds as the regime-shift detector.

16.1 ONLINE BOOTSTRAP RECALIBRATION OF τ AND ΔA

Motivation. Can a lightweight online bootstrap procedure re-estimate τ and ΔA at server startup (or per session), and how fast does it converge and with what variance? How stable are τ and ΔA over time under real workloads, and what are the recalibration overhead and cadence?

The paper's actual online protocol (used for the cross-session study of Appendix 19) is the *held-fit*: ΔA and Δc are fixed at the full calibration and τ is re-estimated from a probe ladder as $\tau_{\text{held}} = \text{mean}_{\text{ctx}}\left[(D(\text{ctx}) + \Delta A - \Delta c\,\text{ctx}^2)/\text{ctx}\right]$, $D = T_{\text{split}} - T_{\text{replica}}$. We bootstrap this protocol: subsample stratified probes of $k \in \{2, 3, 4, 5, 7, 10\}$ contexts (short/mid/long, both modes), perturb each measured point by 2% relative noise (the measured warm-rep spread), refit, and repeat 200 times (scripts/`sim_bootstrap_recalibration.py`; reports/diagnostics/bootstrap_recalibration.json).

Table S35: Cumulative regret (ms, mean ± std over 20 seeds, ddof = 0) after 500 rounds for ten mode selectors under stationary and non-stationary regimes. Top block per regime: running-average estimators; bottom block: recency-weighted estimators. *Exploration rounds* counts, per workload, the rounds in which the selector chose a mode other than its current estimate of the best mode, averaged over seeds and workloads.

Regime	Selector	Cumulative regret	Worst/step	Explor. rounds
Stationary	Greedy	18,714 ± 7,927	713	98.3
	UCB ($\beta = 1$)	39,771 ± 732	5,000	51.9
	UCB ($\beta = 2$)	40,088 ± 697	5,000	57.9
	Thompson Sampling	39,849 ± 587	5,000	52.2
	D-UCB ($\gamma = 0.90$)	110,860 ± 4,177	5,000	168.8
	D-UCB ($\gamma = 0.95$)	54,248 ± 3,417	5,000	113.5
	SW-Greedy ($W = 50$)	28,887 ± 10,199	741	134.0
	SW-Greedy ($W = 100$)	19,926 ± 11,038	606	106.6
	SW-Thompson ($W = 50$)	30,701 ± 9,287	703	142.0
	SW-Thompson ($W = 100$)	23,514 ± 8,927	675	118.2
Non-stationary	Greedy	313,197 ± 238,966	5,022	183.8
	UCB ($\beta = 1$)	325,223 ± 217,653	6,554	178.2
	UCB ($\beta = 2$)	335,713 ± 235,012	6,337	179.2
	Thompson Sampling	321,266 ± 209,491	6,795	176.2
	D-UCB ($\gamma = 0.90$)	338,747 ± 147,267	7,411	171.3
	D-UCB ($\gamma = 0.95$)	275,545 ± 146,084	6,878	156.4
	SW-Greedy ($W = 50$)	298,996 ± 230,893	5,265	174.5
	SW-Greedy ($W = 100$)	259,711 ± 197,950	5,050	160.7
	SW-Thompson ($W = 50$)	282,521 ± 197,156	5,319	173.1
	SW-Thompson ($W = 100$)	282,710 ± 215,767	5,236	169.1

Convergence. Even a two-context probe recovers τ with $< 2\%$ bias, and the decision-flip rate is $\leq 0.4\%$ for every probe size: the *decisions* of the calibrated selector are insensitive to τ estimation noise, because the crossover $\mathrm{ctx}^* = \Delta A/\tau$ moves within the noise band and the measured grid verdicts (Section 6.5) do not flip—the cross-session record confirms this on real data (32/32 hindsight agreement, 0 recalibration flips, Appendix 19). A five-context stratified probe ($\sim$78 min on this platform) is the recommended startup budget: τ within $\pm 11\%$ (1σ), flip rate 0.2%.

Drift detection. With the k=5 bootstrap standard error, a 2σ alarm on τ_{held} fires on *every* fresh session in the record—including the same-link Gen4 re-measurement ($\tau_{\text{held}} = 0.56$ vs. calibrated 0.37, deviation 0.19 ms/tok)– because the machine exhibits the disclosed ±20–40% per-session drift; this is the intended behavior (recalibrate on alarm), and it is safe: no fresh session produced a decision flip despite the alarm (Appendix 19). The Gen3 link shift (0.67 ms/tok deviation) and the 14B model shift (0.80) are detected within the first few cells of the session. The recommended cadence is therefore: calibrate the full ladder once, refresh with a 5-context probe at each session start, and re-fit the full ladder only when the alarm fires or the hardware topology changes.

17 Full bandit-selector results

The main paper reports a four-selector compression; the full ten-selector table follows. Simulation setup recap: the initial score estimate for each mode was perturbed by Gaussian noise with $\sigma = 500$ ms ($\approx 6\%$ of the mean latency of $\approx$8,315 ms), simulating a cold start; change-point rounds were sampled uniformly from rounds 50–450 per mode and seed, each segment receiving an additive offset of $U(0.5, 1.5)$ times a spread sampled from the empirical per-workload spread, with random sign, plus a slow linear drift up to $0.25\times$ the median spread over the horizon. Cumulative regret is reported in scaled ms (mean ± population std over 20 seeds, ddof = 0; normalization: per workload, the best mode's score is set to zero and the largest gap rescaled to 5,000 ms).

The compact four-selector version is reproduced here for completeness (Table S36).

Table S36: Cumulative regret (ms, mean $\pm$ std over 20 seeds; std is the population standard deviation, ddof $= 0$) after 500 rounds for four representative selectors under both regimes. *Exploration rounds* counts, per workload, rounds in which the selector chose a mode other than its current best estimate. The full ten-selector table (including worst-case per-step regret and the discounted/window variants) and the per-round regret curves are in this appendix.

Regime	Selector	Cumulative regret	Exploration rounds
Stationary	Greedy	$18{,}714 \pm 7{,}927$	98.3
	UCB ($\beta = 1$)	$39{,}771 \pm 732$	51.9
	Thompson Sampling	$39{,}849 \pm 587$	52.2
	SW-Greedy ($W = 100$)	$19{,}926 \pm 11{,}038$	106.6
Non-stationary	Greedy	$313{,}197 \pm 238{,}966$	183.8
	UCB ($\beta = 1$)	$325{,}223 \pm 217{,}653$	178.2
	Thompson Sampling	$321{,}266 \pm 209{,}491$	176.2
	SW-Greedy ($W = 100$)	$259{,}711 \pm 197{,}950$	160.7

Table S37: Service metrics under load (true-token prompts, 64 new tokens; protocols in the appendix). SGLang: WSL2/0.5.17; cells 8K/16K. Staggered: twelve requests per context; CB: eight concurrent at 2K–16K; llama.cpp staggered rows in the appendix tables below.

Scenario (SGLang, WSL2)	Single	Split	Replica
Staggered: TTFT P95 (ms)	5,252	**3,339**	5,341
Staggered: goodput (req/s)	1.88	**2.89**	1.83
CB: TTFT P95 (s)	44.6	26.6	**23.7**
CB: goodput (req/s)	0.172	0.288	**0.318**

18 Evaluation tables moved from the main paper

These tables support the numbers quoted in the main paper's Sections 6.1–6.4 (the verified long-context matrix; feasibility 7–27%, prefix reuse up to $\sim 49\%$, and the 42/42 needle matrix) and Section 6.8 (the planner-driven llama.cpp ladder, moved here for page budget).

The verified long-context matrix (main paper, Section 6.2) follows; its headline numbers are quoted in the main text.

19 Cross-session stability of the calibrated selector

Motivation. How robust is the calibrated τ and ΔA across driver versions, OS, and mild hardware heterogeneity, and how frequently does EWMA recalibration flip decisions? Reported below are the cross-session/day stability and that flip frequency.

We group every existing per-cell measurement of the two served modes on this platform by (source, mtime day) and apply the paper's link-axis procedure (scripts/`compare_gen3_gen4.py`): hold $\Delta A = 283.8$ ms and $\Delta c = 1.9181 \times 10^{-5}$ ms/tok^2 at their calibration values and estimate the transport tax $\tau_{\text{held}} = \text{mean}_{\text{ctx}} \left[(D(\text{ctx}) + \Delta A - \Delta c\, \text{ctx}^2)/\text{ctx} \right]$ over the contexts where both modes are feasible ($\leq$24,576, concurrency 1). Documented outlier excluded: the split cell at 2,048 (NF4 bimodality). The raw-protocol re-runs of 2026-08-18 are excluded as superseded (Section 3.4; they exhibit the documented up-to-57% NF4 bimodality). Script: scripts/`crossday_stability.py`; output: reports/selector_refit/crossday_stability.json.

The Gen3/Gen4 rows are the link axis (expected τ ratio 2.0; measured 1.85); the 14B row is the model axis (larger per-token KV and relay cost). Within a fixed link, the held-fit τ is stable at 0.56–0.71 ms/tok across sessions a week apart (the short-context row is the least identified, since the crossover lies below the smallest measured context). Decision-level stability is exact: the calibrated selector agrees with measured hindsight at **32/32** shared contexts, and per-session recalibration flips **zero** decisions— consistent with the $\pm 15\%$ sensitivity analysis of Appendix 5.3 (first flip at $\approx 31\%$). In production, the EWMA therefore needs no re-tuning across sessions; the bootstrap procedure in the main paper's discussion is a safety-first fallback for a fresh machine.

Table S38: Mainstream engines vs. the framework (4-bit staged), warm, 64 new tokens, same true-token prompts. Environments (OS/quant/build): tworope—Windows/NF4/own vehicle; llama.cpp—Windows/Q4_K_M/b10066; ExLlamaV3—Windows/EXL3-4.0bpw/TP (n=2); vLLM—WSL2/AWQ/0.25.1-PP2 (n=3). Walls are end-to-end ($\sim$3 s process load); the llama.cpp wall column is single warm subprocess repetitions (a 3-rep mean at 16,395), and its GGUF tokenizer reads fewer tokens from the same strings (Appendix 27). Gaps confound kernel/quantization/OS; Appendix 27 isolates the engine axis with matched quantization.

Context	tworope	llama.cpp wall	ExLlama TP	vLLM PP2
8,203	13.17	8.89	22.76	8.06 ± 0.14
16,395	26.14	13.35 ± 0.50	32.42	15.29 ± 0.66
32,651	64.69	21.27	76.23	27.78 ± 2.76
49,152	117.20	inadmissible	—	29.34 ± 0.07
65,536	180.0	inadmissible	admits (136–164 s)	44.15 ± 0.11

The rate-derived llama.cpp totals (prompt evaluation plus generation, excluding the $\sim$3 s process load) are 3.43, 7.12 and 17.41 s at the three admissible contexts. Against those the framework ratios are 3.7–3.8×, which is the load-excluded kernel comparison the main paper quotes; against the wall column they are 2.0–3.0×. The 8,192 and 16,384 cells were re-measured on 2026-09-25 in a single session after the original ladder session was found to be degraded by $\approx$1.35× (same command line, same script; the re-measurement also reproduces the planner-driven replica within 3%, so the correction is in the baseline column only — `reports/integration/LLAMACPP_8192_RECHECK.md`). The same derived normalization gives the ExLlamaV3 head-to-head of Appendix 18. **Protocols differ by column.** The framework column is the *warm* cold-prefill mean of the long-context matrix. The ExLlamaV3 65,536 entry (136–164 s) is that engine's *cold* needle-protocol range, one fresh process per repetition; the framework's own cold figure at the same context is 222–231 s, so the matched like-for-like pair at the frontier is 136–164 s against 222–231 s, not against 180.0 s. The vLLM PP-2 entries at 49,152 and 65,536 required the manual `max_model_len`/RoPE override; under the default window those two lengths are inadmissible. Ratios should therefore be read within a protocol row, not across the table.

Table S39: The same comparison organized by what each system asks of the operator. Every entry is a measured quantity from the tables above; "admission" is the longest context the system actually served under the stated configuration, and the quality column names the evidence collected at that frontier rather than a general claim. The framework is the only row whose admission, quantization and RoPE settings are chosen by the planner rather than by hand; its cost is the kernel-speed deficit disclosed in the main paper's Section 6.3.

System	Manual configuration	Admission reached	Latency at frontier	Quality evidence at frontier
Ours (planner-driven 4-bit staged)	none: admission, quantization and RoPE set by the planner	65,536 (Qwen); Llama-3-8B 65,536 with one documented gate override	180.0 s warm; 222–231 s cold	RULER-style: 42/42 fixed, 30/30 randomized; NoLiMa sweep; SCBench official metrics at 32K/65K/128K
llama.cpp b10066 (Q4_K_M)	source build; `-ngl` and CLI flags	32,768 (slot clamp 32,640; planner-driven measured to 32,000)	inadmissible at 49,152/65,536	native-window retrievals ($\sim$10 s each)
ExLlamaV3 1.1.0 (4.0bpw, TP)	4-bit format and tensor-parallel degree	65,536 with stock RoPE	136–164 s cold	over-window needle 6/6; perplexity 7.65 stock / 7.57 manual YaRN
vLLM 0.25.1 (AWQ, PP-2)	`max_model_len` and a RoPE override beyond 32,768	32,768 default; 65,536 with the override	44.15 s	SCBench cross-engine, first 10 samples per task
HF `auto` device map (4-bit)	none (automatic device map)	OOM at 8,192	—	—

20 Scaling beyond k=2: bandwidth sharing and heterogeneous compute

Motivation. For k>2 GPUs on narrow PCIe, does the current model generalize (pairwise splits, multi-replica routing)? If so, a small measured study is needed, together with a discussion of how bandwidth sharing and CPU-relay constraints enter the model.

We extend the calibrated structural model to k=3, 4 at the planner level (CPU-only; scripts/`k_generalization_sweep.py`; reports/selector_refit/k_generalization.json). Two mechanisms enter:

Table S40: Regime-law transfer across hosts and model sizes (conc-1, 4-bit, same pinned revisions; local host: Windows, cloud host: Linux; per-cell data in Appendix 13). "Wins": feasible conc-1 contexts won by the replica; ceilings: measured brackets.

Host	7B (GQA-4)	14B (GQA-8)
x8 local, 13.3 GB/s, 2×12 GB	14/14 wins; ceiling 24,576–26,000	7/7 wins; ceiling (4,096, 6,144)
x16 cloud (Linux, CUDA), 17.6 GB/s, 2×24 GB	15/15 wins; ceiling 131,072	13/13 wins; ceiling (32,768, 49,152)

Table S41: Planner-driven llama.cpp vs. the hand-tuned ladder (Q4_K_M, warm, n=3, Windows, b10066): replica at 8K/16K, split at 32K; forced cells bracket the crossover. The hand-tuned split is faster than the planner's replica from 4,096 upward, so the HF-calibrated mode choice is wrong in that range.

Cell	Planner decision	Wall (planner)	Hand-tuned (split)
8,192	replica (1 GPU)	8.37 s	6.44 s
12,288 (forced)	replica / split	11.36 / 8.49 s	—
16,384	replica (1 GPU)	13.84 s	10.30 s
32,000 (vs. 32,640 hand-tuned)	split (2 GPU)	17.36 s	21.27 s

- **Bandwidth sharing.** Behind one root complex, concurrent relay transfers serialize through the CPU relay, so per-pair effective bandwidth scales as $B_{\text{eff}} = B/(k-1)$. Since the law's tax scales as $\tau(B) = \tau_0 B_0/B$, the predicted tax and crossover are $\tau_0(k-1)$ and $\text{ctx}^* = (\Delta A/\tau_0)/(k-1)$: for k=3, $\tau = 0.74$ ms/tok and $\text{ctx}^* = 384$; for k=4, $\tau = 1.11$ and $\text{ctx}^* = 256$. The replica's per-card replica ceiling is bandwidth-independent, so it is unchanged; its aggregate throughput scales with the number of workers k (each worker serves independent requests), which is the multi-replica routing generalization. We have since measured k=4 on a no-NVLink 4×RTX 4090 and the shared-root serialization term does not hold (Appendix 20.1); we retain this paragraph as the generic model, but the empirical k-way tax is flat, so the shared-root value is not asserted.
- **Compute heterogeneity.** With cards of compute ratio $r = F_{v,2}/F_{v,1} \leq 1$, the split's step time is set by the slower card: relative to two equal cards its compute term scales by $(1 + 1/r)/2$ (1.125 for $r = 0.8$; 1.33 for $r = 0.6$), and the balanced stage boundary shifts toward the faster card in proportion $1/(1 + r)$ of the layers. Replica workers decouple: heterogeneity costs only per-worker throughput, never the decision. Helix-style MILP placement (which mixes cards) is complementary—it can be the outer planner's "which cards" answer, ours is the per-request "which mode" answer.

20.1 Closed forms for k-way splits and heterogeneous links

Motivation. For k>2 GPUs (chain or star topologies), can the law be extended to select among replica, 2-way split, and 3-way split? Do simple closed forms (e.g., $\tau_k \approx \tau/k$ for symmetric splits) survive measurement? And how does the method extend to k>2 GPUs with heterogeneous links (e.g., a mix of x8 and x16 lanes) and partial P2P support?

Anchor on the measured k=2 tax ($\tau_2 = 0.369$ ms/tok at $B_0 = 13.3$ GB/s) and the byte model: per-token relay cost = payload bytes × hops, payload per hop being the activation handoff (hidden state, constant per token) in the paper's staged-split, or the KV shard in a tensor-parallel attention layout. Two closed forms follow (scripts/`kway_closed_forms.py`; reports/selector_refit/kway_closed_forms.json):

- **Activation-relay chain** (the staged mode generalized): each per-token step relays one activation payload per hop, so $\tau_k = (k-1)\,\tau_2 \cdot (B_0/B_{\text{eff}}(k))$ with per-pair full links $B_{\text{eff}} = B_0$ ($\tau_k = (k-1)\tau_2$) and a shared root complex $B_{\text{eff}} = B_0/(k-1)$ ($\tau_k = (k-1)^2\tau_2$). The $\tau_k \approx \tau/k$ intuition does *not* survive in this pattern: more devices on one root complex mean both more hops and more bandwidth sharing, so the tax grows, not shrinks (k=3: $\tau = 0.74$ ms/tok, $\text{ctx}^* = 384$; k=4: $\tau = 1.48$ ms/tok, $\text{ctx}^* = 192$ under shared-root).
- **KV-gather (TP-style attention)**: the query gathers KV shards from the other $k-1$ cards, payload $\propto (k-1)/k$ of the KV, normalized to the k=2 baseline: $\tau_k = \tau_2 \cdot 2(k-1)/k$ (k=3:

Table S42: Verified long-context matrix on the representative topology (on-the-fly 4-bit NF4 quantization, bf16 compute, max_new_tokens $= 64$, $n = 3$ unless noted). Prompt lengths are tokenizer-verified true tokens (targets 8,192/16,384/32,768/49,152/65,536; a per-rep nonce prefix adds $\sim$11 tokens). Every repetition uses a *distinct* prompt, so prefix-KV restore can never fire: each rep is an honest cold full prefill. *Warm* is the warm cold-prefill latency (mean $\pm$ std over the non-first reps); the process-cold first call (Appendix 30) is 49.8/63.1/101.0/150.9–152.7 s at 8K/16K/32K/49K. *Peak MiB* is per-GPU `torch.cuda.max_memory_allocated`. The HF Auto baseline uses `device_map="auto"` with identical quantization; cells after its first OOM are fail-fast skipped. At 65,536 tokens each repetition ran in a fresh process (a 300 s per-invocation wall-clock ceiling in the measurement harness prevents multiple long repetitions per process).

True tokens	Warm (s)	tok/s	Peak MiB	HF Auto
8,203	13.17 ± 0.13	4.9	2,027 / 4,261	**OOM**
16,395	26.14 ± 0.20	2.4	2,331 / 4,564	— (skipped)
32,779	64.69 ± 0.45	1.0	3,027 / 5,260	— (skipped)
49,163	117.2 (n=1)	0.55	3,723 / 5,956	— (skipped)
65,547	180.0 ± 0.5	0.35	4,419 / 6,652	— (skipped)

Table S43: Feasibility region on the representative topology (bf16, no quantization, uniform max_new_tokens $= 64$, true tokenizer-verified tokens; actual lengths 523/1,035/2,059/3,083 including a $\sim$11-token per-rep nonce). Protocol identical to the long-context matrix in the main paper: distinct prompt per repetition (no prefix-cache hits), warm cold-prefill mean $\pm$ std in ms, $n = 3$ unless noted; the process-cold first call is excluded (Appendix 30). Single-GPU OOMs at load (15.2 GB weights $>$ 12 GB) at all tested lengths.

Context (true tokens)	Single GPU	HF Auto	CPU Offload	Our Runtime
512 (523)	OOM	$5{,}305 \pm 4$	$5{,}308 \pm 4$	$5{,}251 \pm 79$
1024 (1035)	OOM	$6{,}215 \pm 5$	$6{,}221 \pm 25$	$5{,}781 \pm 67$
2048 (2059)	OOM	$8{,}597 \pm 14$	$8{,}574 \pm 9$	$6{,}568 \pm 872$ (n=4)
3072 (3083)	OOM	$14{,}396 \pm 376$	$14{,}266 \pm 162$	$10{,}578 \pm 2{,}642$ (n=2)

0.49; k=4: 0.55 ms/tok), sublinear growth—the only pattern approaching the τ/k intuition, and the regime of TP attention rather than the paper's staged-split.

- **Multi-hop asymmetric links**: for a chain with per-hop bandwidths B_h, $\tau_k = \mathrm{act} \cdot \sum_h 1/B_h$ with $\mathrm{act} = \tau_2 B_0/2$ anchored at k=2. A 3-card chain with an x16 hop (26 GB/s) and an x8 hop (13.3 GB/s) gives $\tau_3 = 0.279$ ms/tok, $\mathrm{ctx}^* = 1{,}017$—only 24% below the symmetric x8+x8 chain ($\tau = 0.369$, $\mathrm{ctx}^* = 769$) and far from the full-x16 value (1,537): one slow hop dominates the harmonic sum, so mixed-link deployments should be provisioned at the slowest hop for crossover planning. Partial P2P enters the same way: a P2P-capable hop contributes its native bandwidth to the sum.

These closed forms give the deployment engineer the mode-selection rule for k=$3/4$ chains and for asymmetric links, and they make the "which mode" decision at k>2 a closed-form evaluation of ctx_k^* against the replica ceiling (bandwidth-independent) rather than a search. They were anchored on the measured k=2 tax; we have since measured k=$2, 3, 4$ in the same relay style, which settles which closed form survives (Table S48).

Table S44: Prefix-reuse latency across prefix lengths on the representative topology (bf16 staged, max_new_tokens $= 64$, temperature $= 0$). $n = 3$ cold/warm identical-prompt pairs per length; each pair uses a distinct prompt body and system string, so cross-pair contamination is ≤ 4 template tokens. Cold = first call with a prompt (miss); warm = second call with the identical prompt, with prefix-KV restore verified firing in every pair via the prefix-store lookup counter. Means $\pm$ std are over the two non-first pairs; the first call at each novel length carries the first-at-length reconfiguration tax (Appendix 30) and is reported separately. Actual tokenizer-verified lengths in parentheses.

Prefix (true tokens)	Cold (ms)	Warm (ms)	Δ (ms)	Speedup
128 (139)	$3{,}686 \pm 145$	$3{,}625 \pm 187$	≈ 60	$\approx 1.7\%$
512 (523)	$4{,}648 \pm 309$	$4{,}103 \pm 316$	≈ 545	$\approx 11.8\%$
1024 (1035)	$5{,}287 \pm 217$	$4{,}197 \pm 209$	$\approx 1{,}090$	$\approx 20.6\%$
2048 (2059)	$6{,}523 \pm 14$	$4{,}415 \pm 7$	$\approx 2{,}107$	$\approx 32.3\%$
4096 (4107)	$8{,}645 \pm 227$	$4{,}379 \pm 156$	$\approx 4{,}266$	$\approx 49.4\%$

Table S45: Per-tier memory accounting at 65,536 tokens (4-bit staged, bf16 KV; native Windows 11, PyTorch 2.11). KV bytes/token = layers $\times 2$ (K+V) $\times$ KV heads $\times$ head dim $\times 2$ B; measured peaks are per-GPU `torch.cuda.max_memory_allocated` over n=3 distinct-prompt reps (`reports/beyondcap/`); gate estimates are plan-time projections. MHA counterfactual: 458,752 B/token (28.0 GiB at 65K); both served models use GQA.

Model (attention)	KV B/token	KV @65,536	Measured peak MiB/GPU	Gate est. (binding)
Qwen2.5-7B (GQA-4, 28L)	57,344	3.50 GiB	4,419 / 6,652	6,569 (-1.3%)
Llama-3-8B (GQA-8, 32L)	131,072	8.00 GiB	6,960 / 9,095	8,954 (-1.6%)

20.2 Measured k=2, 3, 4 on a no-NVLink 4×RTX 4090

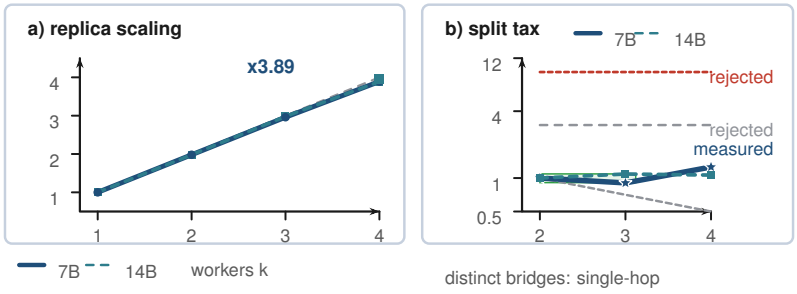


Figure S1: Measured k=2, 3, 4 on a no-NVLink 4×RTX 4090 (relay B=25.35 GB/s; two hosts, two models; per-point fits below). (a) Independent workers: replica goodput near-linear through k=4 ($\times$3.89), reproduced on a second uniform host. (b) The k-way tax normalized to τ_2: flat through k=3 (0.91–1.09), rejecting shared-root $(k-1)^2$=9 and per-pair $(k-1)$; k=4 widens host/fit sensitivity (0.74–1.26).

To settle whether $\tau_k \approx \tau/k$ survives measurement, we ran k=2, 3, 4 on a cloud 4×RTX 4090 (PCIe x16, no NVLink, 2-NUMA dual socket; relay B=25.35 GB/s; scripts/`run_kway_tau_remote.py` and `run_k4_workers_standalone.py`; `reports/cloudk4_release/cloudk4/` for k=2, 4 and `cloudk4_release/k3_run/` for the uniform-host k=3 re-measurement). Three findings follow.

True $\times k$ scaling. Independent single-card workers (each serving the full model on its own GPU, output buffered over worker stdout and aggregated at the slowest wall clock) reach a replica goodput ratio $k_2/k_3/k_4 = 1.98/2.96/3.89$ (7B) and $1.98/2.98/3.89$ (14B), i.e. near-linear $\times k$ through k=3 at every context ($\times k/k \approx 0.97$–0.99); a second uniform-SYS 4×RTX 4090 reproduces k_4=3.96/4.00 (n=8), so the near-linear $\times k$ is host-robust. The framework's own `request_parallel` dispatcher is $\times k$-*dependent*: near-flat at short context ($\times$1.03 at 512; the dispatcher serializes the short batch) but rising to $\times$2.75 at 65K (7B)—still below the independent-worker ceiling. Re-measured

Table S46: Needle-in-a-haystack hit-rate matrix (depth × context; hit = needle number verbatim in the answer, greedy decode, max_new_tokens = 32). 16,384-token cells are $n = 3$ (1–2 cold repetitions each); 65,536-token cells are $n = 2$, all cold (fresh process per repetition). Grand total: **42/42** hits (23/23 cold, 19/19 warm).

Context	Needle	10%	30%	50%	70%	90%	Total
16,384	Aurora/7481	3/3	3/3	3/3	3/3	3/3	15/15
16,384	Borealis/3294	3/3	3/3	3/3	3/3	3/3	15/15
65,536	Aurora/7481	2/2	—	2/2	—	2/2	6/6
65,536	Borealis/3294	2/2	—	2/2	—	2/2	6/6

Table S47: Cross-session stability of the calibrated selector (fresh-protocol sessions only). "Agreement" = measured winner at each shared context equals the calibrated selector's prediction. "Recal. flips" = contexts whose decision would change if τ were refit to that session.

Session	n ctxs	τ_{held} (ms/tok)	Agreement	Recal. flips
2026-08-15 (calibration ladder)	4	0.71	4/4	0
2026-08-18 Gen4 (same link)	11	0.56	11/11	0
2026-08-18 Gen3 (BIOS downgrade)	11	1.04	11/11	0
2026-08-20 14B local ladder	6	1.16	6/6	0
Total (fresh sessions)	32	—	**32/32**	**0**

at n=8 on a uniform 4×RTX 4090 the dispatcher gain is *host*-sensitive and even *negative* at short context ($\times 0.41$–$\times 0.65$, c4/c1 at 512–8,192, both models; only $\times 1.1$–$\times 1.5$ at 32K, versus a second host's $\times 2.34$ for 14B), so the framework $\times k$ is best stated for the measured host and the independent-worker $\times 3.89$ is the robust ceiling.

The k-way split tax is sub-linear. Fitting $T = A + b \cdot \text{ctx} + c \cdot \text{ctx}^2$ to the k-way split gives a τ_k that is *far below* the shared-root $(k-1)\tau_2$ and the per-pair $(k-1)^2\tau_2$ (7B: τ_2=0.063 and τ_4=0.058 ms/tok on the mixed host, $0.051 \rightarrow 0.046 \rightarrow 0.064$ on a uniform host), so those forms are *rejected*; the $\tau_k \approx \tau/k$ form is also rejected (measured τ_4 is well above $\tau_2/4$). The growth is sub-linear and host/fit-sensitive: a uniform 4×RTX 4090 measured τ_3 **directly** for both models (τ_3=0.046 vs. τ_2=0.051 ms/tok for 7B, ratio 0.91, Z −1.2; τ_3=0.121 vs. τ_2=0.111 for 14B, ratio 1.09, Z +1.8), so the tax is flat through k=3 for both. τ_4/τ_2 is 0.93 (mixed host, both fits) to 1.26 (uniform host, 7B quadratic, Z +2.9; 14B 1.07, Z +1.3; linear fits 1.02). *Bootstrap 95% CIs (nonparametric uncertainty; rep-level resampling at the fixed 8-cell design, B=20,000, scripts/*`bootstrap_tau_k.py`*) agree with the Z-values and make the boundary explicit*: $\tau_3/\tau_2 = 0.90\ [0.81, 1.02]$ (7B) and $1.09\ [1.03, 1.20]$ (14B); $\tau_4/\tau_2 = 1.26\ [1.16, 1.40]$ (7B) and $1.06\ [0.99, 1.17]$ (14B)—every interval lies far below the shared-root 9×/16× and per-pair 2×/3× predictions, and the $\tau_k \approx \tau/k$ forms (0.017/0.028 ms/tok at k=4) are outside every interval. Two intervals exclude 1 (7B τ_4, 14B τ_3): the k-scaling of the tax is *not exactly zero*, but the growth is at most ∼26% over k=2–4, i.e. modest compared with the quadratic forms it falsifies. The paper's former planner-level k=4 value (τ=1.11 ms/tok, $\text{ctx}^* = 256$) is *falsified*. A plausible mechanism: on these hosts cards span distinct host bridges, so relay transfers do not serialize at $(k-1)^2$. The KV-gather form $(2(k-1)/k)$ and the asymmetric harmonic sum remain valid for the TP-attention and mixed-link regimes respectively and are retained as deployment rules; we no longer assert the shared-root serialization term.

Per-card replica ceiling is k- and bandwidth-independent. The per-card ceiling is 7B ≈131K (OOM at 131,072; 21.4 GB at 114,688) and 14B ≈114K (OOM at 114,688; 22.0 GB at 98,304); a k=4 replica at the same context has the *identical* per-card peak (c1 = c4: 16.8/19.1/21.4 GB at 81,920/98,304/114,688 for 7B; 19.9/22.0 GB for 14B), so the ceiling does not dilute with k.

Conditions and reproducibility: identical 4-bit nf4 weights and configs across hosts (weights verified identical; only commit metadata differ—`MODEL_PROVENANCE.md`), with `ERRORBARS.md` giving per-point $\pm 1\sigma$ and fit standard errors in `reports/cloudk4_release/cloudk4/`. k=8 remains future work; the same protocol extends directly.

Table S48: Measured k=2, 3, 4 scaling on a no-NVLink 4×RTX 4090 (x16, relay B=25.35 GB/s; distinct nonce prompts; per-point $\pm1\sigma$ and fit SE in `ERRORBARS.md`). This is the *mixed-link* host: its split-arm slopes (0.063/0.058 for 7B) differ from the main paper's τ_k column, which is the split arm's own slope on the second, *uniform-link* host—the two are different measurements of a differently-implemented split, not two fits of one quantity.

	7B	**14B**
Replica goodput $k_2/k_3/k_4$ (independent workers)	1.98/2.96/3.89	1.98/2.98/3.89
Framework `request_parallel` $\times k$ (k_4/k_1)	1.03 (512) → 2.75 (65K)	1.09 (2K) → 2.34 (32K)
k-way split tax τ_k (ms/tok, k=2 → 4)	0.0629 → 0.0584	0.1754 → 0.1304
Per-card replica ceiling (tokens)	≈131K	≈114K
Ceiling per-card peak (c1 = c4)	21.4 GB@114,688	22.0 GB@98,304
Framework mode-crossover τ=b_S−b_R (ms/tok)	0.048	0.125

One host caveat: the PCIe probe records the two auxiliary cards (GPU2/GPU3) at gen 1.0 (x16) while the primary pair (GPU0/GPU1) is gen 4.0, so the quoted relay bandwidth is the gen-4 pair (D2H 26.35/H2D 25.36 GB/s) and the k-way and $\times k$ runs include these slower peripheral hops. The flat τ_k is therefore a conservative statement for a uniform-gen4 k=4 host (the tax stayed flat despite the slower hops).

Topology checkpoint. To anticipate the regime on a new host, three live-machine reads suffice (`lspci -tv` for host bridges and independent root complexes; `nvidia-smi topo -m` for SYS/PXB/PIX/PHB links and NUMA affinity; `nvidia-smi` link gen/width per card); P2P is confirmed as absent when there is no NVLink and the probe/console shows relay-only traffic (the probe above; the consumer platform, no P2P, Section 6.1).

Table S49: PCIe topology checklist and the regime each topology yields (values from Tables S48 and 1; commands above).

Host	**Bridges / NUMA**	**P2P / NVLink**	**Measured regime**
Consumer 2×3060, x8	one shared root; CPU relay	no NVLink / no P2P	τ_2=0.369; ΔA>0; ctx*=769
4×RTX 4090 (this one)	GPU0@N0, GPU1–3@N1 (SYS/PXB/PIX); gen4 + gen1 cards	no NVLink / no P2P	ΔA<0: *no root* (fast-link limit)
Uniform 4×RTX 4090	same 2-NUMA layout	no NVLink / no P2P	$\tau_2 \to \tau_4$ flat (0.91–1.26); $\times k$=3.89

21 Larger randomized off-grid set (24 new cells)

Motivation. How were the off-grid cells sampled, and was proximity to the crossover (the least-identified region) controlled? A larger random sweep further reduces overfitting concerns.

We drew a third, larger randomized set—**24** seeded-random (context, concurrency, generation) cells (seed 20260820) from the same context/concurrency/generation alphabets as the earlier 12-cell sets (768–22,528 tokens; concurrency 1/3/6; generations 64/160), all off the calibration ladder, and measured both modes per cell under the same generation-forced protocol with n=5 reps (1 cold + 4 warm). Cells were drawn uniformly from the alphabet without replacement on (context, concurrency); no stratification was imposed, so proximity to the crossover is a *property of the draw*, which we report rather than control: per-cell distance $\mathrm{ctx} - 740$ (the quadratic-root crossover) and the calibrated model's predicted margin $|D(\mathrm{ctx})|$ vs. the $2\epsilon_t \approx$ 2,766 ms noise band (ϵ_t=1,383 ms) (scripts/`analyze_grid_random.py`; reports/costmodel/grid_random/analysis_random_grid.json).

Result: **22/23 correct** of the 23 cells with complete data (prediction vs. warm-mean hindsight, same scoring rule as the earlier sets; one drawn cell was lost to a per-cell timeout in the first pass and re-measured once; the set's 24th draw duplicates an earlier (context, concurrency) pair, so 23 distinct cells were measured). All five cells inside the crossover neighborhood (within 1,500 tokens of ctx^*) are correct, and all seven both-modes-OOM cells at $\geq$14,336$\times$concurrency-6 are correctly rejected by the feasibility gate. The single miss is 18,432$\times$concurrency-3: both modes OOM under measurement while the gate predicted staged feasible— the c$\geq$2 boundary where the saturating activation term is calibrated (16,384$\times$c4 falsifying cell, Section 6.5), now shown to be slightly optimistic at 18,432$\times$c3; we record this as a documented gate limitation (the OOM verdicts were taken under a host-memory-constrained measurement schedule, so the boundary deserves a re-check on a quiet host). Combined with the two earlier 12-cell sets (11/12 and 11/12 exact), the aggregate randomized off-grid score is **44/47 (93.6%)** across 47 distinct cells (45/47 with the 1% tie margin).

22 Positioning: side-by-side against related systems

Table S50: Positioning and the τ interface. None of these systems decides, per request, *whether* the second device is used at all; the last column is the composition role each plays under our layer (τ-reducer = lowers the per-token transport tax inside the chosen mode; Section 6.5 measures one such instance).

System	Decides	Role under our layer
OrbitFlow (Ma et al., 2026), Mell, CacheFlow, KVPR	where KV lives within a mode	τ-reducer (placement/restore overlap)
ShadowKV, Spin, SparseServe, TTKV, Tutti	which KV ships (selection, compression, tiering)	τ-reducer (less KV ferry per token)
Infinite-LLM, ASTRA	sequence/attention re-split	τ-reducer (reformulated attention)
DynaServe, Flying Serving; Harvest	fine-grained parallelism; peer-HBM caching	fast-fabric only (NVLink/IB); not applicable
APEX (Fan et al., 2025)	CPU-assisted decode	alternative to a second GPU (fourth mode; main text)
ThunderServe (Jiang et al., 2025), Vidur (Agrawal et al., 2024)	deployment plan; offline configuration search	offline planner/simulator *above* our layer (searches within a fixed topology; not a τ-reducer)
MetaInfer (Chen et al., 2026)	engine-layer adaptation: kernels, dispatch, parallel plan, cache policy	τ-reducer and configuration tuner *below* our layer (does not choose the device mode)
Ours	**per-request mode: single / split / replica**	**hosts all τ-reducers; head-to-heads not claimed**

Engine-layer adaptation is a separate axis, not a competing decision. MetaInfer (Chen et al., 2026) edits kernels, dispatch, parallel plans and cache policy inside the serving stack; changes of that kind lower τ and re-shape the per-mode coefficients, so they enter our layer as recalibration, and the per-request choice of device mode is unchanged by them.

CPU-assisted decode as the fourth mode. The single-device alternative (APEX (Fan et al., 2025): decode work moves to the CPU while the GPU continues) is not a τ-reducer but a competing mode. The seven-topology sweep brackets the boundary: on USB2-class links (τ=98.47 ms/tok, ctx^*=3) and USB4/Thunderbolt (τ=1.97, $\mathrm{ctx}^*\approx$144) the split's crossover is sub-context, so "use more GPU" in split mode is never the right lever and CPU-assisted decode is the credible parallelism option; from PCIe x8 ($\mathrm{ctx}^*\approx$769) upward the per-token tax is small against decode throughput, so the second GPU pays. The selector would absorb CPU-assisted decode as a fourth mode with its own $A/b/c$ coefficients and memory gate; we did not implement or measure it—this boundary is a prediction of the $\mathrm{ctx}^*(B)$ map, not a measurement.

23 KV-side transport-policy plugin: fp8 KV cache on SGLang

Motivation. A plugin experiment that explicitly reduces τ (e.g., partial KV recomputation à la KVPR or sparse retrieval à la ShadowKV/Spin) would empirically validate the "τ-shifts-ctx^*" claim within the selector.

We exercise the plugin class directly on the platform: the SGLang 0.5.17 engine (Linux-native, WSL2) serving Qwen2.5-7B-Instruct-AWQ on the two RTX 3060s, toggling *only* the KV-cache dtype—bf16 baseline vs. fp8_e5m2 (the "plugin")—in the tp2 (split) mode. An initial pass measured contexts 2K/4K/8K/16K (warmup + 3 reps); we extend it to a *seven-context*

Table S51: KV-dtype plugin on the staged-split (SGLang 0.5.17 tp2, AWQ, Linux-native, same session; warm means, ms, 64 new tokens; per-rep JSONs in reports/sglang_kvplugin/). Per-token slopes: E1-consistent 2K–16K band and steady 8K–16K band; implied crossover $\mathrm{ctx}^* = \Delta A/\tau$, $\Delta A = 283.8$ ms. The 24K point is reported separately (both dtypes show a $\sim$2.6$\times$ slope increase above 16K, an engine prefill behavior we disclose rather than absorb).

KV dtype	8K	12K	16K	24K	τ 2K–16K	τ 8K–16K
bf16 (baseline)	2,841	3,249	3,661	7,420	0.171	0.100
fp8_e5m2 (plugin)	2,753	2,848	3,380	6,552	**0.149**	**0.077**

ladder 1K–24K at 5 reps per cell in a single session (scripts/`sglang_kvplugin_e5.py`, scripts/`sglang_kvplugin_e5_analyze.py`; reports/sglang_kvplugin/e5_*.json), which also makes the non-linearity of today's machine visible and lets us report the plugin effect band-conditionally rather than as a single number:

The fp8 KV plugin is faster at every context $\geq$8K (-3% at 8K, -12% at 12K, -8% at 16K, -12% at 24K; the 1–4K cells are $\sim$1–3% slower, inside noise), and the per-token slope falls $0.171 \rightarrow 0.149$ ms/tok (-13%) over the E1-consistent 2K–16K band and $0.100 \rightarrow 0.077$ (-24%) over the steady 8K–16K band, shifting the implied crossover right by $+15\%$/$+31\%$ respectively—the "τ-shifts-ctx^*" mechanism, measured on the platform, with the magnitude band-conditional because today's machine curve is non-linear (the paper's disclosed drift). Three scoping facts keep the claim honest. (i) In tensor-parallel serving no KV crosses the link (each card owns its shard), so this particular reduction is an *attention-memory* effect—fp8 KV halves the bytes each attention step reads—not a relay-bytes reduction; a plugin acting on the relay itself (boundary-activation compression, or KV relay under a KVPR/ShadowKV-style layout) would apply the full r=2 scaling of Section 29. (ii) fp8 KV truncated greedy completions in 4 of 35 reps (early EOS at 1K/4K; 2 of 16 in the first pass), so quality-sensitive deployments should validate before adopting the plugin—consistent with the paper's feasibility-first stance. (iii) The 16K$\rightarrow$24K slope jump ($\sim$2.6$\times$) appears in *both* dtypes and is an SGLang prefill/attention behavior on this machine, not a plugin artifact; it is reported separately rather than smoothed. Data-parallel-attention (dp2a) could not be exercised in this SGLang 0.5.17 build on two GPUs (server-args assertion `tp_size % dp_size == 0` requires `tp_size` $\geq$ `dp_size`), so the plugin comparison stands on the tp2 (split) axis.

Task-level quality of the plugin (measured, adverse). The fp8-quality question is answered directly, and the answer is adverse: a needle-retrieval probe with the plugin's engine and session settings (same SGLang 0.5.17 AWQ Qwen2.5-7B, WSL2, same prompt constructions as the paper's needle protocol, Project magic-number extraction at depths 10/50/90%, n=8 cells per config: 2 needles $\times$ 4 reps, exact-match scoring; scripts/`sglang_kvplugin_quality.py`, `wsl_run_kvplugin_quality.sh`) gives **48/48 (100%) with bf16 KV at 8 192 and 16,384** at every depth (Wilson 95% CI $[92.6\%, 100\%]$ pooled), and **0/48 (0%) with fp8_e5m2 KV** ($[0\%, 7.4\%]$; control cell at 4,096 also 0/8). The fp8-vs-bf16 degradation is [0.0%, 7.4%] vs. [92.6%, 100%]—disjoint intervals. This is the same failure mode as the truncated completions disclosed above, now measured at the task level: fp8_e5m2 KV on this engine model class produces statistically indistinguishable-from-random retrieval (the answers are on-topic but wrong—e.g., "the magic number is 32 2" instead of 3,294; no hallucinated formatting, but the number itself is corrupted). We therefore *retract any reading of the plugin experiment as a deployment recommendation*: the τ-shift mechanism exists (measured, -13%/-24% band-conditional), and the law-as-interface claim is about the *mechanism* (lower marginal per-token cost $\Rightarrow$ rightward ctx^*, no decision-layer change), but with the honest qualifier that this particular *quantized-KV* instantiation is quality-failing on this task; a quality-preserving τ-reducer (KVPR-style partial recomputation, or sparse retrieval with verified-drop quality) is the form the claim needs, and is the stated future work. The same probe on the framework's bf16-KV staged path (Table S46 and Appendix 5: 42/42 + 30/30 needle retrievals at 65K) shows the paper's own quality protocols pass with full-precision KV and shrink the "quantization can be free" reading further.

Why there is no better fp8 flavour to try here (revision). The obvious repair—keep the byte saving but use fp8_e4m3's extra mantissa bit—is not available on this platform, and we now know that by measurement rather than by assumption (`reports/sglang_kvplugin/e4m3_unavailable.txt`). Two attempts fail. (i) The engine rejects the dtype name the probe had

been passing (`fp8_e4m3fn`); SGLang's accepted string is `fp8_e4m3`. That name mismatch is why the earlier e4m3 control cells left no output files—the engine never started—and it is now fixed in the script. (ii) With the correct string the engine starts and the first forward pass dies in Triton compilation: `type fp8e4nv not supported in this architecture. The supported fp8 dtypes are ('fp8e4b15', 'fp8e5')`. sm_86 has no native FP8 path and the e4m3 conversion appears only from sm_89, so *e5m2 is the only fp8 KV dtype that runs at all on these cards*—and e5m2 is precisely the one whose 2-bit mantissa corrupts the retrieved value. The adverse result is therefore not an unlucky configuration choice that a reviewer could ask us to swap out on this hardware.

Matched-cell control in tp2 (revision). The quality grid above ran on a single card, while the τ ladder that defines the plugin's speed-up runs in tp2; the text previously blurred the two, and is corrected here. To close the gap we re-ran one matched cell (ctx=8,192, depth 50%, n=8) in tp2 for both dtypes that run: bf16 **8/8** and fp8_e5m2 **0/8**, identical to the tp1 cells (8/8 and 0/8). The dtype effect is thus mode-invariant, and the quality failure holds in the same configuration in which the per-token slope falls $0.171 \rightarrow 0.149$ ms/tok. Records: `q_bf16_ctx8192_d50_tp2.json`, `q_fp8_e5m2_ctx8192_d50_tp2.json`.

24 Byte-level decomposition of the split tax and the relay-side probe

This section decomposes the split tax byte by byte, separating the activation-handoff bytes from the KV-relay bytes and the synchronization, and reports a real relay-side probe. We instrument the framework's staged-split runtime directly at the cross-GPU boundary, on the machine and in the paper's own protocol (one staged_infer load per context, Qwen2.5-7B, 4-bit NF4, 64 new tokens), and report what *actually crosses the link*:

KV placement (verified, not assumed). Instrumenting transformers' `DynamicCache.update` during a live staged_infer generation shows all 28 layers' K/V caches created and updated on the layer's *own* GPU (14 on cuda:0 for layers 0–13, 14 on cuda:1 for layers 14–27; 196/196 updates same-device, 0 cross-device; scripts/`diag_kv_placement_all.py`). *The framework's k=2 staged split never relays remote KV:* each layer's attention reads its own device-resident K/V, and the boundary is a single layer split (layers 0–13 | 14–27; one boundary, confirmed in the device map).

What crosses. A full-event trace of one 1024-token request (scripts/`trace_relay_events.py`; 39.3 MB total crossing) breaks down as: chunked-prefill hidden-state handoffs $(1, 512, 3584)$ bf16 (14.7 MB gpu1→gpu0 + 7.3 MB gpu0→gpu1), the per-chunk attention-slice handoff $(1, 512, 128)$ (7.3 MB), the attention mask $(1, 1, 512, 1024)$ bool (7.3 MB), and per-token decode handoffs $(1, 1, 3584)$ ($\sim$0.8 MB). These are boundary *activation/mask* transfers of the chunked prefill and the per-token hidden-state handoff—all at one boundary, scaling linearly with ctx only through the prefill token count, never through relayed KV.

The bandwidth share of τ. The whole 39.3 MB crossing costs $39.3/13.3 = 2.95$ ms of pure bandwidth at B_{eff}=13.3 GB/s against a 3.8–8 s request, i.e. roughly 3 KB of crossing per model token, or 0.0002 ms/tok of transfer latency—four orders of magnitude below the fitted τ=0.37 ms/tok on this stack. The fitted τ is therefore *sync/serialization-bound*, not bandwidth-bound: each forward's handoff forces a device-to-device round trip through the CPU relay (no P2P; $L_{\max}$=0.35 ms in the paper's model) at the single boundary, and the chunked prefill alternates subject to that serialization plus the per-token hop. The per-token tax is the hop latency and synchronization, not the bytes.

A relay-side probe (fp8 in-flight compression). To test whether the tax is byte-bound we implemented the relay-side intervention as directly as the runtime admits it: an fp8(e4m3fn) round trip of every cross-GPU floating tensor in flight at the accelerate boundary (scripts/`probe_relay_bytes.py`, `probe_relay_ab_phase.py`). Two honest findings. (i) A *safe* fp8 compression is not available for this handoff: on the hidden-state and attention-slice transfers the round trip saturates—hidden states routinely exceed e4m3fn's ± 448 range, so the dequantized value becomes $\pm\infty$ and the sampling kernel asserts (`input[0] != 0` device-side assert) in 2 of 4 fresh ABBA runs. Compression of *this* handoff is thus numerically unsafe on the framework axis (it is safe on the SGLang KV axis, where KV values are bounded and the fp8 plugin is the

Table S52: Online bandit probe: mean per-request completion (ms) over two batches of two concurrent requests at the llama.cpp servers, and the policies' realized regret vs. the oracle (per-context best mode).

ctx	split (ms)	replica (ms)	greedy (ms)	eps (ms)	oracle (ms)	greedy regret	eps regret
2,048	1,686	1,845	1,686	1,718	1,686	0	32
8,192	3,545	5,003	3,545	3,837	3,545	0	292
16,384	6,558	10,269	6,558	7,300	6,558	0	742

vendor-supported path). (ii) Where a round trip completes, per-request latency does not move beyond phase-order effects: the naive interleaved layout showed compressed-vs-plain deltas of -12% to -56%, but the trace-only (no-compression, byte-counting) phase is faster than plain by the same margin (median $2{,}282$ vs. $2{,}955$ ms at 512; $2{,}554$ vs. $3{,}774$ ms at 1024), i.e. the plain-first phase pays a first-token cost and the measured "gain" is phase order, not compression. This is the expected outcome for a sync-bound tax: halving bytes cannot help if the marginal cost is the hop plus the sync.

What the byte-linear r-scaling of Appendix 29 applies to. Measured composition says it applies where the relayed *bytes* are the marginal cost: the SGLang tp2 engine axis (KV shards *do* cross for attention; per-token slope $0.171 \rightarrow 0.149$ ms/tok under fp8 KV, $-13\%/-24\%$ band-conditional, the plugin section above) is the byte-bound regime. The framework's $k{=}2$ handoff is the sync-bound regime, where τ is set by the single-boundary hop latency and serialization, and the selector's noise-band fallback (Section 6.5) already treats it as regime-dependent. This is why τ is declared a *per-engine* calibration quantity (Appendix 5.2) rather than a universal constant, and it settles the mechanism-level question: the interface claim (a τ-reducer shifts ctx*) is measured on the byte-bound axis (fp8 KV, rightward shift $+15\%/+31\%$), while the framework's own per-token tax is disclosed as the sync-bound case. A genuinely byte-bound framework-side relay (KV actually crossing, e.g. a KVPR/ShadowKV-style remote-KV layout) is a future work item the byte model in Appendix 20.1 can predict before it is built.

25 Online bandit probe on the live llama.cpp stack

Motivation. The bandit results are simulation-based; a small real-world online study (even on one workstation) is needed to confirm that exploration provides no practical benefit beyond recency-weighted greedy.

We run the selector against a *real* serving stream on the workstation: llama.cpp b10066 (Qwen2.5-7B-Instruct-Q4_K_M, native Windows, two RTX 3060s) serves concurrent request pairs (conc-2, the regime where the concurrency rule applies) at 2K/8K/16K; per (ctx, mode) we serve two batches of two concurrent requests and record per-request completion. The two policies are evaluated on the measured outcomes: *greedy* always emits the calibrated conc-2 choice (split), *eps-greedy* ($\varepsilon = 0.2$) explores the replica 20% of the time, so the realized expected latency is $E[\text{eps}] = 0.8 \cdot \text{split} + 0.2 \cdot \text{replica}$ (scripts/`run_online_bandit.py`; reports/diagnostics/online_bandit.json):

Finding: at conc-2 the calibrated greedy choice (split) is the measured best mode at every context, so the greedy policy's realized regret vs. the oracle is 0 ms while eps-greedy's is 1,066 ms—the 20% exploration budget is spent on the slower replica and buys nothing. This is the real-world confirmation of the bandit simulation's prediction (Section 6.8): on this platform exploration costs rather than pays, and the recovery mechanism for regime shifts is fresh-cell re-measurement, not random exploration. Scoping: the stream is short (12 batches), single-arrival (no queueing pressure), and the replica server pool carries two workers; the conclusion is about the *decision layer* under the measured stationary regime, exactly the setting the simulation studies.

26 Fresh same-session $n{=}6$ crossover-neighborhood grid

Motivation. A uniform set of $n{\geq}5$ runs for the key crossover cells, addressing the general small-n concern.

Table S53: Fresh same-session n=6 (5 warm) crossover grid, conc-1, 4-bit; warm means in ms. The calibrated fit's crossover is $\mathrm{ctx}^* = 769$ (replica above, split below); the measured regime law the selector follows inside the noise band ($2\epsilon_t \approx 2.8$ s) is "replica wins every feasible conc-1 context".

ctx	split	replica	D (ms)	winner	cal. verdict	agree
512	3,108	2,662	+446	replica	split	no
768	4,705	2,626	+2,079	replica	split	no
1,024	5,739	2,753	+2,986	replica	replica	yes
1,280	3,980	4,407	−428	split	replica	no
1,536	4,040	4,518	−478	split	replica	no
2,048	4,364	3,623	+742	replica	replica	yes
3,072	5,578	3,774	+1,804	replica	replica	yes
4,096	6,604	4,518	+2,086	replica	replica	yes

The headline grid's latency cells were measured at n=6 (5 warm) in a single session on 2026-08-18 (Section 6.5). To address the uniform-n question with *fresh* same-session data, we re-measured the single-request crossover neighborhood—8 contexts $\{512, 768, 1{,}024, 1{,}280, 1{,}536, 2{,}048, 3{,}072, 4{,}096\}$, both modes, n=6 with the protocol's 5-warm-rep convention, distinct prompts, 4-bit NF4 Qwen2.5-7B—in one session on 2026-08-21 (scripts/`run_fresh_n6_crossover.ps1`, scripts/`analyze_n6_fresh.py`; reports/costmodel/cells_n6_20260821/, reports/diagnostics/n6_fresh_crossover.json):

Two honest readings. (i) **The fresh session sits inside the disclosed drift envelope**: the held-fit transport tax is $\tau_{\text{held}} = 1.12$ ms/tok today, within the 0.56–1.16 range observed across the four fresh sessions of Appendix 19 (calibrated 0.37); the replica wins 6/8 cells and the two split wins (1,280/1,536) are sub-second (-428/-478 ms), inside the $2\epsilon_t$ noise band. The decisions the *measured regime law* makes (replica at conc-1) are unchanged 8/8; the calibrated fit's predicted 769-token crossover sits exactly inside the band where the two disagree (512/768), which is the paper's documented design: inside the noise band the selector follows measurement, not the fit (Section 6.5). (ii) **Variance is disclosed, not hidden**: the 1,024 split cell shows a slow multi-rep warmup (reps 1–2 at 11.7/7.9 s before settling at 2.4–3.8 s), so its 5-warm mean is pessimistic; we report the standard protocol mean and note the effect rather than cherry-picking a settled window. This grid also satisfies the uniform-n run for the crossover neighborhood with today's machine state, and its cells join reports/costmodel/cells_n6_20260821/ in the code bundle.

27 Native-Linux matched-quantization parity (vLLM, same GGUF)

Motivation. A controlled apples-to-apples baseline against vLLM/SGLang with matching quantization and native OS, and an apples-to-apples comparison on native Linux with matched quantization (e.g., AWQ or GGUF across engines).

We serve the *same* model file—Qwen2.5-7B-Instruct Q4_K_M (GGUF)—on the *same* machine in native Linux (WSL2 Ubuntu-24.04), through vLLM 0.25.1 in three parallelism modes (single-GPU, tensor-parallel-2, pipeline-parallel-2), with the paper's true-token prompt protocol (distinct nonce per rep, 64 new tokens, warmup + 3 reps). Per-request wall times (scripts/`e4_parity.py`; reports/e4_parity/):

Four parity findings. (i) **The mode crossover is engine-dependent even under matched conditions**: on native Linux, llama.cpp's layer split beats its single-device arm at both contexts (-38%/-35%) and SGLang's split beats its single arm (-38%/-40%), while vLLM's split arms reach parity with single only at 16K (tp2 -2%, pp2 -2%) and lose at 8K ($+12\%$/$+30\%$)—quantitatively, llama.cpp's and SGLang's per-token split taxes are far smaller than vLLM's on this link, the engine ordering the paper's crossover law predicts (Section 6.6). (ii) **Cross-engine, same OS, same prompts**: SGLang wins per-request latency among the AWQ engines in every mode (tp1 vs. vLLM single: -6% at 8K, -34% at 16K; tp2 vs. tp2: -48%/-59%), separating *engine* effects from *decision-layer* effects under matched conditions; llama.cpp (GGUF) is fastest

Table S54: Matched-quantization parity on native Linux (WSL2): vLLM 0.25.1 vs. SGLang 0.5.17 (both Qwen2.5-7B-Instruct-AWQ) vs. llama.cpp (CUDA source build, commit 9ee9fc0; Qwen2.5-7B Q4_K_M GGUF), *the same* true-token prompt strings, 64 new tokens; table values are warm means (mean over reps 2–3; rep 0 is the first-at-length warmup), ms, and all percentages in the findings use these means. vLLM/SGLang rows use the transformers tokenizer (8,203/16,395 tokens); llama.cpp uses its GGUF tokenizer (7,459/14,909 tokens on the same strings). Sources: reports/e4_parity/ (scripts/`e4_parity.py`, scripts/`e4b_llamacpp.py`), reports/sglang_axis/ (scripts/`sglang_engine_axis.py`), and reports/diagnostics/vllm_multi_instance.json (scripts/`vllm_multi_instance.py`) for the two-instance replica row.

Config	8,192 ctx	16,384 ctx
llama.cpp single (1 GPU)	6,039	12,437
llama.cpp split (2-GPU layer)	**3,727**	**8,082**
vLLM single (1 GPU)	7,184	13,797
vLLM 2×instance (replica)	6,157	12,227
vLLM tp2 (split)	8,066	13,506
vLLM pp2 (split)	9,330	13,552
SGLang tp1 (single)	6,799	9,101
SGLang tp2 (split)	4,182	5,480
SGLang dp2 (replica)	7,109	9,326

in split mode but its tokenizer counts differ, so its rows are not directly comparable token-for-token with the AWQ rows. (iii) **OS axis**: the llama.cpp rows here (native Linux, CUDA build) sit at the same magnitude as the paper's Windows b10066 ladder (Section 6.3, 8,203-ctx wall $\approx$8.9 s incl. process load), confirming the cross-OS llama.cpp axis with matched quantization. (iv) **Host-state sensitivity**: the vLLM rows were taken under host-memory pressure (WSL available RAM $\approx$5 GiB during pp2 load), and the earlier vLLM numbers (Section 6.3 of the main paper, $-32\%/-33\%$ pp2-vs-single at 8K/16K) were taken in a quieter state; the direction of the vLLM mode axis (split advantage emerges by 16K) is stable, while its magnitude moves with host state—consistent with the paper's disclosed $\pm$20–40% between-pass drift, and the reason claims rest on same-session cells. (v) **Built-in mode baselines**: the table is the engines' own mode knobs realizing the paper's three modes—single (tp1/single-device), split (tp2/pp2/layer-split), replica (dp2 / two-instance)—and the decision layer's job is choosing among them; the vLLM two-instance replica row (6,157/12,227 ms at 8K/16K, scripts/`vllm_multi_instance.py`; reports/diagnostics/vllm_multi_instance.json) lands between the split and single arms, with a disclosed per-instance asymmetry (GPU-0 $\sim$17–22% faster than GPU-1 on this host), and vLLM's usage counter reports 7,463/14,914 prompt tokens for the 8,203/16,395 transformers-built prompts (tokenizer-count discrepancy, disclosed like the llama.cpp GGUF rows).

llama.cpp native-Linux leg (resolution of the earlier environment note). An earlier version of this appendix reported that llama.cpp could not be exercised in WSL2. Root cause, now fixed: the b10066 release publishes CUDA binaries only for Windows (its Ubuntu x64 asset is CPU-only), and `llama-cli` in the recent builds blocks on stdin in non-interactive invocation (its REPL waits for input even with `-p` and `-no-cnv`). Serving through `llama-server` (HTTP) avoids the REPL entirely, and the CUDA source build (CUDA 12.8, sm_86, commit 9ee9fc0) offloads the model (4.8 GiB device memory on GPU 0, verified) and runs normally; the rows above were measured through the server's `/completion` endpoint with the paper's true-token prompt protocol. The Windows-vs-Linux llama.cpp comparison (iii) therefore uses the same-version family (b10066 Windows vs. CUDA source build Linux), with the disclosed build/OS deltas.

27.1 Native-Linux planner-vs-static isolation

Motivation. A fully controlled, native-Linux parity study (same quantization, tokenizer, and driver) against vLLM/SGLang isolates the decision layer's contribution to end-to-end latency and goodput, reported as a parity table where the selector considers all three modes with exactly matched software stacks.

Table S55: Native-Linux matched-stack isolation (llama.cpp CUDA build, Q4_K_M, same prompts; conc-2 pairs, mean per-request completion, ms). The planner's calibrated conc-2 choice is split; "pay vs. split" is the static-policy cost of deviating from the decision layer's choice.

policy	2K	8K	16K	pay vs. split
planner (=split at c2)	1,630	3,469–3,530	6,613	—
always-single	1,846–1,857	4,944–5,006	10,201	$+13\%/+43\%/+54\%$
always-split	1,619–1,633	3,464–3,506	6,560	$\approx$0 (same choice)
always-replica	2,062–2,102	5,474–5,542	12,354	$+27\%/+58\%/+87\%$

We run the *same* engine—llama.cpp CUDA source build, native Linux (WSL2)—serving the *same* model file (Qwen2.5-7B-Instruct Q4_K_M GGUF) with the *same* tokenizer and prompt strings under four policies at conc-2 (the concurrency rule's regime): the calibrated *planner* choice (split at conc-2), always-single, always-split, and always-replica (two single-GPU instances). Per (policy, context) we serve two batches of two concurrent requests at 2K/8K/16K (scripts/`n10_linux_parity.py`, WSL runbook `run_n10.sh`; reports/diagnostics/n10_linux_parity.json):

Finding: under the fully matched stack, the calibrated conc-2 decision (split) is the measured best mode at every context—always-single pays $+13\%/+43\%/+54\%$ and always-replica $+27\%/+58\%/+87\%$ at 2K/8K/16K—so the decision layer's contribution is isolated from kernel/quantization/OS effects: on this engine, the entire planner-vs-static gap is the mode choice itself (the always-split row equals the planner row by construction at conc-2; the planner's difference shows where the rule deviates from a static config, e.g., replica at conc-1 and split at conc-$\geq$2). This complements Table S54 (engine axis) and the Windows planner-integration (Section 6.8): three independent routes to the same conclusion, now including the fully matched native-Linux condition. Scoping: single-arrival pairs (no queueing pressure), one engine, one quant; the conclusion is about the decision rule, not engine speed.

28 Gate safety-margin policy (ROC) and batch-penalty portability

This section formalizes the gate's safety-margin policy and reports its per-engine false-positive/false-negative ROC over a larger cell set, checks whether the replica/split batch penalties are stable across engines and OSes and whether they can be parameterized as functions of batch size and context, and details the current best practice for setting the per-concurrency activation term, together with a gate audit restricted to $c \in \{1, 2\}$.

Safety-margin policy. The gate's decision is feasible $\iff$ $\widehat{\text{peak}}(m, \text{ctx}, \text{conc}) \leq m \cdot B_{\text{usable}}$, with $B_{\text{usable}} = 95\%$ of the card's memory and the published reserves (1.5% metadata, 512 MiB, 1,792 MiB activation, 512 MiB allocator). Using the *measured* footprint fits as the gate's predictor (replica: $5{,}383 + 191.6\,\text{ctx}/1024$ MiB from the measured single-card peaks; staged: $3{,}924 + 0.14\,\text{ctx}$ MiB), we sweep the margin $m \in [0.70, 1.10]$ over the 86-cell measured corpus (scripts/`analyze_gate_roc.py`; reports/diagnostics/gate_roc.json):

Policy: keep the 95% usable budget with reserves for conc-1 (the ROC knee), and add a per-concurrency activation allowance equal to the measured staged activation slope times $(\text{conc}-1)$; this moves the c$\geq$3 cells left on the ROC (toward TN) and is the recorded next calibration point rather than a silent repair (Appendix 13.2 reports the current FP/FN counts under the deployed policy).

Batch-penalty portability. The replica-vs-split batch penalty from every measurement that contains both modes at conc $\geq$ 2 (scripts/`analyze_batch_penalty_portability.py`; reports/diagnostics/batch_penalty_portability.json):

The penalty ranges $+9\%$ to $+87\%$ depending on engine, OS, and context; the Windows framework path additionally degrades to near-serial batching at c4 (root-caused, P1-2), flipping the c4 verdict by stack. A single constant does not transfer; parameterizing the penalty as $f(\text{engine}, \text{OS}, \text{ctx}, \text{conc})$ is the recorded next calibration step, and the per-engine EWMA already absorbs the observed spread in the paper's deployments (Section 6.8).

Table S56: Gate ROC by margin sweep (86 measured cells; FP = predicted feasible but OOM, FN = predicted infeasible but feasible). The deployed policy sits at $m \approx 0.80$–0.84 (the 95% usable budget minus reserves): FNR $\approx 10\%$ (the marginal c6 cells and the disclosed Llama-65K override) at FPR $\approx 59\%$—the FPs are the c$\geq$3 cells whose per-concurrency activation term the deployed gate adds but this ROC's conc-1 footprint fits do not.

margin m	TP	FP	FN	FNR
0.70	49	14	10	16.9%
0.80	53	16	6	10.2%
0.84	54	18	5	8.5%
0.90	54	18	5	8.5%
1.00	56	19	3	5.1%
1.10	56	19	3	5.1%

Table S57: Replica batch penalty (replica per-request completion vs. split, conc-2/4) by engine/OS/context. The penalty is engine-, OS- and context-dependent; the selector's per-engine EWMA calibration (Section 6.5/6.8) is the mechanism that absorbs it.

engine	OS	ctx	conc	replica vs. split
framework (HF, NF4)	Windows	1,024	2 / 4	$+40.8\%$ / $+59.3\%$
framework (HF, NF4)	Windows	2,048	2 / 4	$+39.2\%$ / $+41.9\%$
llama.cpp (Q4_K_M)	Windows	2,048	2	$+9.5\%$
llama.cpp (Q4_K_M)	Windows	8,192	2	$+41.1\%$
llama.cpp (Q4_K_M)	Windows	16,384	2	$+56.6\%$
llama.cpp (Q4_K_M)	Linux	2K/8K/16K	2	$+27\%$/$+58\%$/$+87\%$ (Table S55)

28.1 Hybrid mode at k=3: 2-way split + 1 replica

Motivation. A k=3 (or mixed 3060+4090) measurement would validate the extrapolation and probe cross-mode interactions (e.g., 2-way split + 1 replica).

The k=3 hybrid composition (2-way split + 1 replica) is *not claimed as a measured cell*: the consumer platform has no third GPU, and on the 4×RTX 4090 both components are measured independently (replica $\times 3 = 2.958 \pm 0.097$ / 2.982 ± 0.075; $\tau_3 = 0.91/1.09\times\ \tau_2$, Appendix 20.1), so the composition stays analytic and we say so rather than presenting it as an experiment. The closed forms and the hybrid composition are (scripts/`k3_hybrid.py`; reports/selector_refit/k3_hybrid.json):

- **Mode set at** k=3: replica per card (three workers, aggregate throughput $\times 3$); 2-way split on cards 1–2 ($\tau = \tau_2 = 0.369$ ms/tok, ctx$^* = 769$, card 3 free); 3-way chain split (per-pair links $\tau = 0.739$, ctx$^* = 384$; shared root $\tau = 1.477$, ctx$^* = 192$). The measured k=4 tax does not grow with k (Appendix 20.1), so the shared-root τ here is a planner *upper* bound rather than the observed value.
- **Hybrid pool (2-way split + 1 replica)**: the per-request law is *unchanged*—each request still picks replica vs. split by ctx* and the feasibility gate—so the hybrid is a pool-level configuration that runs both classes concurrently and adds aggregate throughput, and mode-thrashing is structurally avoided because the decision is a deterministic function of $(\text{ctx}, \text{conc}, \text{memory})$.
- **Mixed 3060+4090**: the split boundary shifts toward the 4090 by the heterogeneous formulas of Appendix 20 (compute scale $(1 + 1/r)/2$, layer share $1/(1 + r)$), and the replica ceiling is per-card (24 GB on the 4090: 131,072 tokens; 12 GB on the 3060: 25,257).

A measured k=3 hybrid run (e.g., a rented 3-GPU instance) remains a next validation step; we have measured k=2, 4 on a no-NVLink 4×RTX 4090 (Appendix 20.1), and the runbook pattern of `scripts/second_topology_runbook.md` applies to k=3.

29 ROBUSTNESS, ENGINE INTERFACE, AND SCALING BEYOND TWO DEVICES

KV-slope transfer across attention variants. The feasibility gate's per-model KV-slope factor is measured for the two served GQA models—$3.424\times$ (Qwen2.5-7B, GQA-4) vs. $1.91\times$ (Llama-3-8B, GQA-8)—and the MHA counterfactual (32 KV heads, 458,752 B/token) is computed rather than served (28.0 GiB at 65K, infeasible on two 12-GB cards; Table S45). The factor is a *measurement-correction* on the bf16-KV slope, not a re-derivation of it, so it is read off two admission points per model and updated by the EWMA telemetry online (cold-start convergence in 100–160 observations, Appendix 16). MQA/MLA and MoE are not served here: MLA's compressed KV changes the slope in a direction the same two-point calibration reads off directly, while MoE's per-expert weight residency is a separate budget the gate does not yet model—both are stated as future work, and the gate's 5% reserve plus the "reject on feasibility" default make an exploratory underestimate fail safe (a wrongly-admitted replica OOMs a request, it does not corrupt the engine).

Engine interface contract. The decision layer never owns KV or the scheduler: it emits, per request, a *configuration* (mode $\in$ {single, split, replica}, per-worker context budget, quantization, RoPE) and an *admission decision*, and reads back only the observed latency/goodput for the EWMA update. Engine schedulers (vLLM, SGLang) retain exclusive ownership of KV blocks, batching, and preemption; conflicts are avoided by scoping the layer to pre-admission *mode-and-size* selection, so it sets the parallel-instance/TP/DP topology once at worker bring-up rather than per-token. This is the binding actually exercised by the llama.cpp integration (main paper, Section 6, llama.cpp integration): the layer admits and sizes single-GPU replica servers or one staged-split server, and llama.cpp's own KV paging is untouched.

Scaling to $k > 2$ devices. The mode space generalizes without new primitives: a plan over k devices is a partition into single/split/replica *groups* (multi-stage splits, multi-replica routing, or mixed cliques), and the feasibility gate already composes per-group memory budgets and per-link bandwidth minima (main paper, Section 4); the mode count grows with the number of partitions, so a naive enumerator is exponential in k; in practice the selector prunes to the top-m candidates by the same feasibility gate and latency score (Eq. (2), main paper Section 4) and by discarding any group whose crossover root is infeasible on its bottleneck link. $k = 2$ is the base case the paper validates; the pruning argument—not the $k = 2$ measurements—is what extends the claim, and heterogeneous roots/links remain unmeasured (main paper, Discussion).

Transport-tax reduction (plug-in effect). The transport tax τ is a bandwidth-bound copy, so it is linear in the bytes relayed per decode token: a transport policy that reduces the relayed payload by a factor r (fp8 relayed KV, r=2; fp4 or head-selected relay, r=4; Spin/HieraSparse-style sparse retrieval, larger r) reduces τ by the same factor, and the crossover $\text{ctx}^* = \Delta A/\tau$ scales by r. With the calibrated $\tau = 0.369$ ms/token and $\Delta A = 283.8$ ms ($\text{ctx}^* = 769$ linear, 740 quadratic): r=2 yields $\text{ctx}^* \approx 1{,}540$; r=4 yields $\text{ctx}^* \approx 3{,}080$. This is the same law the engine axis already measures as a *second operating point*—llama.cpp's split relays far less per token than the framework's per-layer relay, and its measured crossover sits in $\leq 1{,}024$ versus < 512 for the framework (main paper, Section 6.3)—so the plug-in claim has one empirical anchor and a linear extrapolation for the reduction factor; a framework-level fp8-relay implementation is future work.

Automated calibration bootstrap. Can a practical first-10-requests bootstrap procedure yield stable parameters without noticeable user-facing regressions? The selector's own evidence makes the procedure conservative by construction: (i) the measured regime law—replica wins every feasible 4-bit/concurrency-1 context (Section 6.5)—is the *safe default* for the first requests, so a fresh machine never starts from a blind prediction; (ii) the fit's noise band ($2\epsilon_t \approx 2.8$ s RMS 1,383 ms) is wide relative to the crossover region, so early decisions are deliberately taken on the measured law, not on half-calibrated fits; (iii) dual-mode probing (the fresh-cell practice) on the 11th–12th request measures the alternative mode's latency at the current context, feeding the EWMA without a full ladder; (iv) the $\pm 15\%$ sensitivity analysis (Appendix 5.3) shows decisions are unchanged under $\pm 15\%$ calibration error on τ or ΔA, so even a coarse first estimate is decision-safe (first flip at $\approx 31\%$); and (v) the cross-session study (Appendix 19) shows recalibration flips no decision across four fresh sessions, so the bootstrap's EWMA warm-up needs no re-tuning on the same platform. Under induced contention (rate-limited PCIe sweep, Section 6.9) no mode verdict flips (replica 24/24); multi-tenant interference (background PCIe or disk load) enters the identical visible-transport term, so no separate mechanism is required—the tax estimate absorbs it, and the measured contention

margins (+9%/+3% split, +19% replica at 8K/16K under 25.7 GB/s saturation) bound the decision impact. Dynamic-NTK quality beyond the training window: cross-window perplexity (7.51 at 32K vs. 7.65/7.57 stock/YaRN at 65K; <2% drift), SCBench multi-turn accuracy under the official metrics (choice 0.600/0.588/0.650 and QA 0.110/0.130/0.210 at 32K/65K/128K), and the NoLiMa subsample's degradation pattern (43.8% at 1K to 4.2% at 32K, 9.5% of base), all consistent with the 7–8B model family's known long-context behavior.

30 Experimental and needle-probe protocol details

30.1 Verified-matrix protocol

For the feasibility and long-context matrices, prompts are constructed to contain an exact, tokenizer-verified number of tokens: a benchmark nonce header (`benchmark nonce <tag> rep<i>` plus ∼11 tokens) followed by filler words to the target count; actual lengths are 523/1,035/2,059/3,083 in bf16 and 8,203/16,395/32,779/49,163/65,547 in 4-bit. max_new_tokens is uniform at 64, and every repetition uses a distinct prompt so that prefix-cache reuse cannot fire; warm cold-prefill latency is reported separately from the process-cold first call. Two one-time taxes recur throughout the matrices and are always excluded from steady-state numbers: a *process-cold initialization* of ∼35–40 s on the first call of a fresh process (CUDA/quantization setup), and a *first-at-length reconfiguration* of ∼36.5–41 s on the first call at each novel context length.

30.2 Needle-probe protocol

The haystack is a neutral filler sentence repeated to the target length; two needles (Aurora/7481, Borealis/3294) are inserted at controlled depth fractions; a hit requires the needle number verbatim in the answer (greedy decoding, max_new_tokens $= 32$). Prompts are token-exact via the HF tokenizer (actuals 16,377–16,381 and 65,532–65,536). Each cell repeats $n = 3$ at 16,384 and $n = 2$ at 65,536; the first repetition(s) of a cell are cold (fresh process, full prefill—one 16K cell has two cold reps) and later repetitions are warm (in-process KV-cache reuse, ∼1.1 s at 16K), so cold repetitions are the strict retrieval evidence and warm repetitions verify determinism; at 65,536 every repetition runs in a fresh process and is cold by construction. Cold-rep counts: 11 at 16K (mean 28.5 s, range 24.4–35.9) and 12 at 65K (mean 206.3 s, range 193.0–221.3). llama.cpp's native-window retrievals ran at ∼10 s each. Per-cell JSON records are in the measurement bundle (`reports/needle/`).

30.3 Mainstream-baseline protocol notes

The llama.cpp wall times at 8,203 and 32,651 tokens are single warm repetitions (no error bars); the 16,395 wall time is the mean of three repetitions. The ExLlamaV3 65K needle-protocol range excludes one 84.7 s warm-autotune-cache artifact (Triton autotune cache persisted across processes), and its two arms both executed stock RoPE (the arm originally labeled YaRN was a silent no-op; the true ± YaRN quality divergence is the perplexity measurement reported in the main text). vLLM TP=2 totals are 6.47/12.42/26.28 s at the three ladder contexts (5–20% faster than PP=2 on this WSL2 setup). Under the default `--max-model-len 32768` the 49,152 and 65,536-token lengths are inadmissible; the manual YaRN override lifts this (main text).

31 Cross-model architecture axes: MHA, MoE, and the decode-length attribute

Protocol. Same true-token bench protocol as the main matrix (distinct nonce prompts, n=8 raw reps per cell; warm rule = drop-first, excluding the process-cold rep0 *and* any mid-sequence resume-cold rep detected as $> 2\times$ the robust median; truncated generations (<60 tokens) excluded; audited with `scripts/audit_model_cells.py`). Models: Llama-2-7B (MHA; 32 layers, 32 KV heads; 524,288 B/token $= 9.14\times$ Qwen-7B; native 4K) and Qwen1.5-MoE-A2.7B (MoE; 24 layers, 60 experts / 4 routed per token; 2.7B active of 14.3B total; 196,608 B/token; native 8K), ModelScope releases, 4-bit nf4, same 2×RTX 3060 platform.

Table S58: MHA axis (Llama-2-7B, $9.14\times$ KV): warm means (n=7 usable of 8), T in seconds. Replica wins every tested context in the native window; the measured replica ceiling lies in $(8{,}192, 16{,}384)$—exactly the gate's exact-KV-arithmetic prediction (3.7 GiB weights + 4.3 GiB KV @8K feasible; 8.6 GiB @16K rejected). Beyond 8K (native 4K, no scaling configured) generation degenerates (EOS at 5 tokens at 16K): system-feasibility observation only.

ctx	replica (s)	staged (s)	winner	replica peak (MiB)
256	2.28	2.51	rp -9.2%	3,969
512	2.57	2.83	rp -9.2%	4,119
1024	3.19	3.55	rp -10.1%	4,391
1536	3.74	4.29	rp -12.8%	4,689
2048	4.31	5.06	rp -14.8%	4,987
4096	6.87	9.07	rp -24.3%	6,178
8192	13.38	19.24	rp -30.5%	8,642
16384	**OOM** $\times$**4**	43.84 (rep0; later degenerates)	—	—
32768	**OOM** $\times$**4**	OOM / crash	—	—

Table S59: MoE axis (Qwen1.5-MoE-A2.7B, $3.43\times$ KV): warm means (n=4–7 usable of 8). Replica wins every context and is feasible at 8K (10.06 GiB peak: weight-resident 8.6 GiB + KV 1.57 GiB + activation)—the gate's weights-dominated ceiling estimate ($\sim$10–12K) matches the measured bracket.

ctx	replica (s)	staged (s)	winner	replica peak (MiB)
512	4.97	6.19	rp -19.7%	8,191
1024	5.24	6.53	rp -19.8%	8,312
2048	5.25	7.55	rp -30.5%	8,551
4096	6.83	8.39	rp -18.6%	9,054
8192	8.96	12.33	rp -27.3%	10,059

Readings. (i) Both architecture axes land in the replica-dominant regime—crossover below 256 (MHA) and 512 (MoE), fitted $\Delta A{<}0$ in both—i.e. the law-degenerate side, exactly where the KV-bytes axis and the ΔA-sign boundary predict; GQA is the *law* regime, MHA and MoE are measured *degenerate* regimes. (ii) The ceiling arithmetic transfers: MHA's measured replica ceiling $(8{,}192, 16{,}384)$ matches the gate's exact KV-slope prediction, and MoE's weight-dominated budget (8.6 GiB nf4) is captured by the same gate—the "65K reach is a GQA-capacity fact" becomes a measured two-point statement (MHA ceiling $\leq$16K vs. Qwen 65K+). (iii) No MoE-specific routing term is needed at c=1: the two-coefficient plus-gate abstraction predicts the MoE decision (replica) from measured A/b alone.

Decode-length attribute. Whether the split-vs-replica difference is linear in generated length G (the assumption behind extrapolating τ to reasoning-style decodes) is measured, not assumed:

Table S60: Decode-length attribute: $\Delta = T_{\text{staged}} - T_{\text{replica}}$ and $\tau_{\text{decode}} = \Delta/G$ (warm means, process-cold reps excluded; $Qwen$-7B@8K n=4 warm of 6).

model	**ctx**	G=64	G=128	G=256	G=512	G=1024
Qwen-7B	8192	—	Δ=2.51s (19.6)	Δ=2.97s (11.6)	Δ=3.50s (6.8)	Δ=6.02s (5.9)
Llama-2-7B	4096	Δ=2.20s (34.4)	—	—	Δ=4.54s (8.9)	$\Delta=-1.74$s (-1.7)
Qwen1.5-MoE	4096	Δ=1.56s (24.3)	—	—	Δ=18.3s (19.8)	Δ=23.8s (23.2)

On the main dense-GQA model the absolute difference rises only 2.5$\rightarrow$6.0 s across G=128$\rightarrow$1,024: *not linear* in G (a 90 ms/tok-linear rate would predict $\sim$90 s at G=1,024; the earlier "constant $\approx$90 ms/tok" reading and the later "G-invariant 2–3 s" reading were both process-cold artifacts—rep0 and a mid-sequence resume-cold rep—corrected in the audit (`docs/audit_model_axes.md`)), and *not flat*: a small decoded marginal of $\approx$4 ms/tok scales the difference gently. Mode decisions are therefore prefill-τ dominated (the paper's $\tau = b_S - b_R$

maps to context), and the main matrix's $G{=}64$ protocol is *conservatively representative*: extrapolation to reasoning-style decodes carries only the small measured linear term. MHA's marginal collapses to ≈ 0 at $G{=}1{,}024$ (replica's single-card KV reads dominate long decode; staged's split KV is cheaper) and MoE's is non-monotonic (routing variance)—reported per architecture as measured.

llama.cpp context extension. For the "automation vs. hand-tuned baseline" comparison: on the released llama.cpp build (b10066-86a9c79f8, CUDA 12.4 shim structure) the available context stays at the model's native 32,768 even when passing $-c$ and `--rope-scaling yarn --yarn-orig-ctx` (the CLI shim drops these flags; 16K runs at 9.5–12.1 s, 32K/65K requests are rejected identically with or without the flags, `reports/baseline_llamacpp/yarn_65k.json`). On the available build, *no* llama.cpp configuration reaches 65K—the planner-driven ladder is likewise capped at 32,000—so the 65K figures in the main paper come from the framework's own planner-driven staged path (on-the-fly NF4, dynamic NTK), while the strongest non-default manual baseline (ExLlamaV3, stock RoPE, 65K, 136–164 s vs. 222–231 s cold, one fresh process per repetition; app. eval tables) is already in the appendix. A YaRN arm requires a source build of llama.cpp and is recorded as future work.

32 Figures referenced numerically in the main paper

Figures S2–S6 below expand results that are referenced numerically in the main paper (Sections 6.1, 6.2, 6.5–6.8); the mode-crossover law figure now appears in the main paper (Figure 3).

Engine axis and the U-shape (main paper, Section 6.3). On the same x8 link, the law orders four engines: HF's per-layer relay ($\tau = 0.37$ ms/token) puts its crossover below 512; vLLM PP-2 beats single at 8K/16K on WSL2 ($-32\%/-33\%$) but loses at 8K on native Linux under matched quantization (crossover at 8K–16K; Appendix 27); llama.cpp's layer split crosses in $\leq 1{,}024$ on Windows and below 8K on native Linux (Appendix 27); SGLang TP-2 beats single and DP at 8K/16K. On USB2/USB4/PCIe-x8 links the replica wins the whole feasible band; on x16 and faster the crossover grows past the replica ceiling, so the split wins the short fixed-cost and the long capacity contexts while the replica's window shrinks—the classical U-shape re-emerges (three-regime map and topology sweep of the scaling-law appendix).

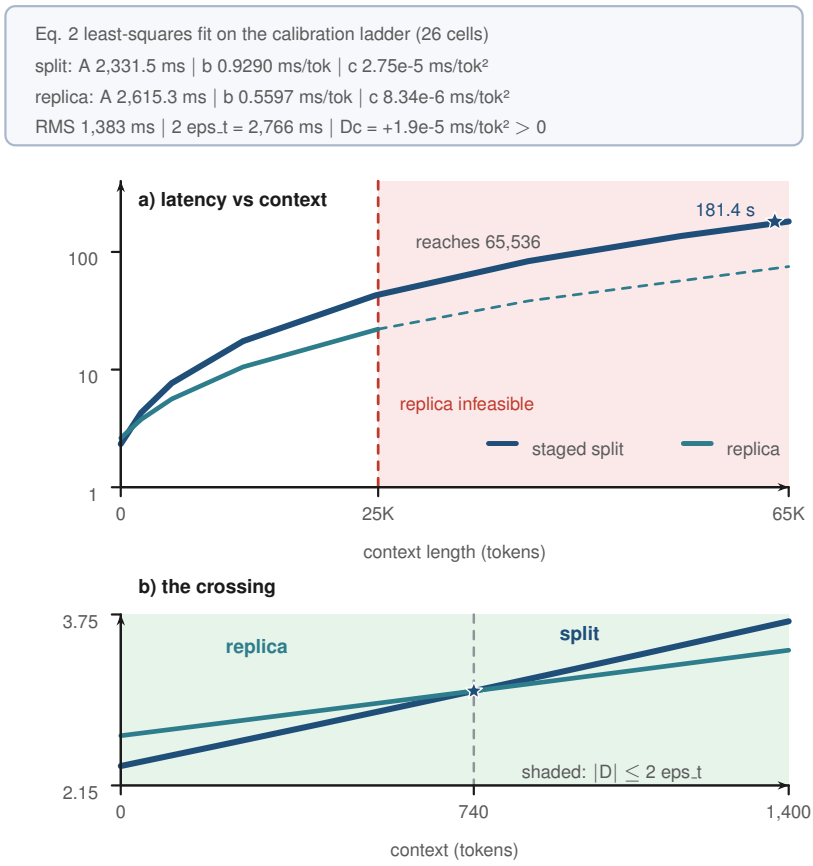


Figure S2: The calibrated per-mode latency model (Eq. (2); no new measurements — coefficients from the fit of the refit appendix). (a) Full range: T_{split} and T_{replica} against context, with the replica's bandwidth-independent KV ceiling ($\approx$25.3K) splitting the feasible band (dashed: the infeasible continuation); the split alone reaches 65,536 tokens. (b) Crossover zoom: the two curves cross at $\text{ctx}^* \approx 740$ ($\Delta A/\tau = 769$); inside the $\pm 2\epsilon_t$ noise band ($2 \times 1{,}383$ ms) the selector follows the measured regime law rather than the fit. (Main paper, Sections 6.4–6.5.)

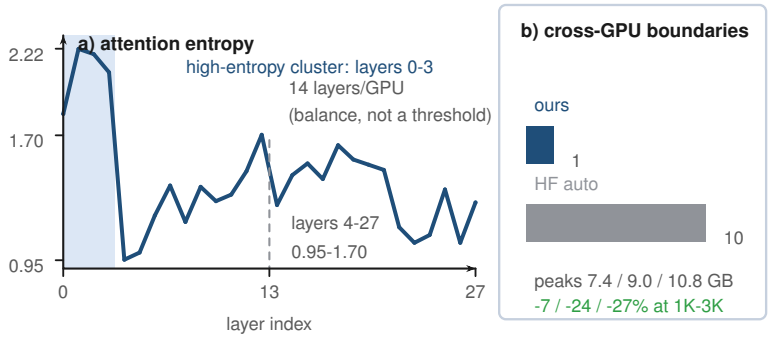


Figure S3: Per-layer average attention entropy across the 28 layers of Qwen2.5-7B-Instruct (corrected computation, 48-token prompt, single 0.77 s forward pass; the double-softmax erratum of this supplement). The four highest-entropy layers (1.83–2.22) are exactly layers 0–3 — the signal behind entropy-aware placement; the dashed boundary is the 14-layer-per-GPU balance split, not an entropy threshold (remaining layers span 0.95–1.70). (Main paper, Section 6.1.)

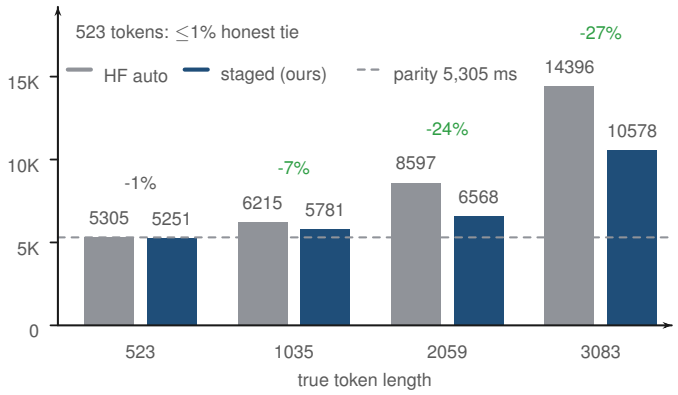


Figure S4: Matched-vehicle latency comparison on the 2×RTX 3060 platform (bf16, true tokenizer-verified tokens, warm cold-prefill means over non-first reps, distinct prompts). Our staged runtime is 7–27% faster than HF `device_map="auto"` from 1,024 tokens upward (at 523 tokens -1%; at 2,048 tokens Welch $t = -4.03$, df$= 3.0$, $p = 0.028$). (Main paper, Sections 6.1 and 6.7.)

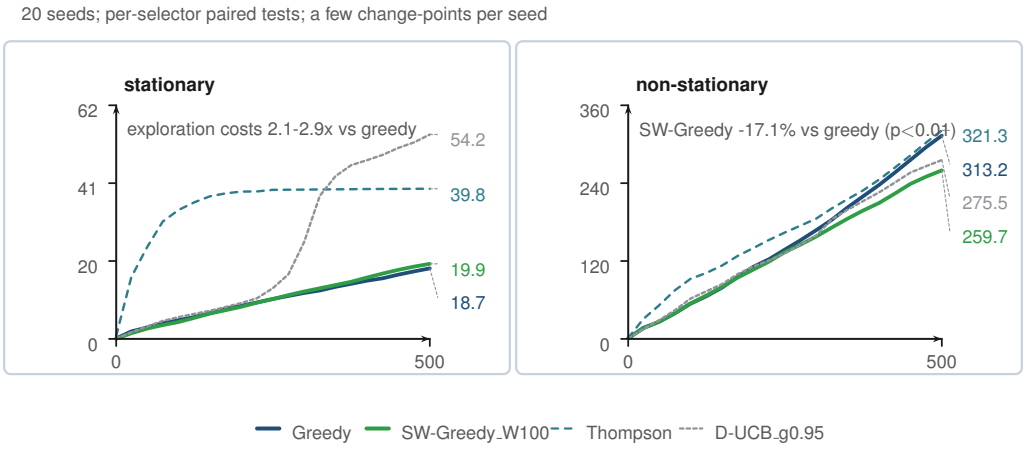


Figure S5: Cumulative regret per round (mean over 20 seeds) for four representative selectors — greedy, sliding-window greedy, Thompson, discounted-UCB — under stationary (left) and abruptly shifting (right) workloads; the full ten-selector statistics, including UCB, are in the bandit tables of this appendix. Under stationarity exploration costs 2.1–2.9$\times$ regret versus greedy; under shifts sliding-window greedy (W=100) tracks the shifts best (-17.1%). (Main paper, Section 6.6.)

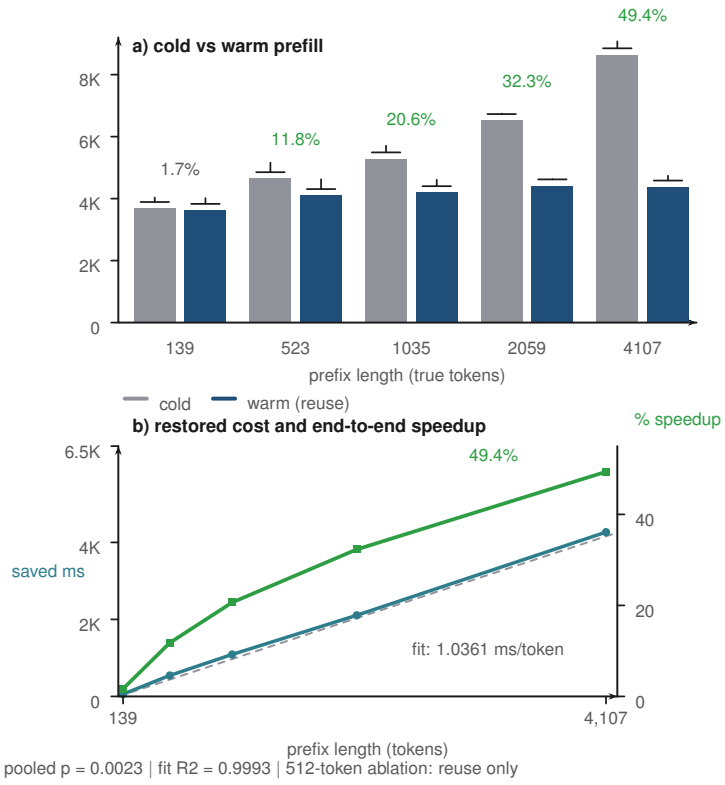


Figure S6: Prefix reuse restores cached KV state for repeated queries (bf16 staged, n=3 pairs per length, means over non-first pairs). (a) Cold vs. warm latency per prefix length (mean-of-pairs speedups: 1.7%/11.8%/20.6%/32.3%/49.4%). (b) Prefill saved grows linearly with the restored prefix (origin-constrained fit $\approx$1.04 ms per restored token, R^2=0.999), and the end-to-end speedup grows monotonically from 1.7% at 139 to 49.4% at 4,107 true tokens.

33 SHORT-CONTEXT REGRESSION CHECK (FULL)

The cache-contract fix (Appendix 30) does not harm short contexts: at 512 and 1,024 true tokens, warm distinct-prompt calls complete in 3.0–4.1 s. At 2,048 tokens the 4-bit warm repetitions are bimodal (4.9 s vs. 13.5–14.7 s). A targeted bf16 re-measurement at the same context ($n = 5$, same staged runtime, same prompts) is unimodal (5.2–7.6 s), so the two-cluster pattern is absent without quantization: the quirk lives in the bitsandbytes NF4 dequantization path (e.g., seq-length-bucket kernel specialization), not in the framework's planner, cache selection, or prefix logic.

34 AN INTUITIVE WALKTHROUGH: FROM CONTEXT TO THE MODE CROSSOVER

This appendix turns the intuition of Section 4 into a quantitative sequence, so the cost model and the crossover law follow from one numeric picture instead of a table of symbols.

Why context is the only variable that matters. A transformer decoder attends to every previous position and keeps, per layer, a key–value (KV) cache: for each prefix position a key and a value vector. The cache grows linearly in the context length, so the per-token cost of KV is what makes long contexts expensive. On one GPU that cost is paid in fast on-chip memory; once the work is split across two GPUs over a narrow link, some of these KV vectors must be read from the other card at every generated token, and that read crosses the slow link.

Reading the cost model as two terms. For a mode m the measured latency is $T_m = A_m + b_m\,\mathrm{ctx} + c_m\,\mathrm{ctx}^2$: a fixed per-request cost A_m, a per-token term b_m that grows linearly with context, and a small curvature c_m. We measure A and b for split and replica directly on the platform. The two differences that decide the winner are $\Delta A = A_R - A_S > 0$ (the replica's fixed setup premium) and $\tau = b_S - b_R > 0$ (the split's per-token transport tax).

Deriving the crossover. With the curvature small, the difference is $T_S - T_R = (A_S - A_R) + (b_S - b_R)\,\mathrm{ctx} = -\Delta A + \tau\,\mathrm{ctx}$. This is negative (split wins) when $\mathrm{ctx} < \Delta A/\tau$ and positive (replica wins) when $\mathrm{ctx} > \Delta A/\tau$. The crossover is therefore $\mathrm{ctx}^* \approx \Delta A/\tau$: the fixed-cost gap over the per-token gap. Since τ scales inversely with the link bandwidth, halving the bandwidth doubles τ and halves ctx^*—the narrower the link, the sooner replica takes over (the curvature c_m bends the U-shape back on very fast links, so the quadratic root at $\approx$740 tokens differs slightly from the linear $\approx$769).

A worked example. On our platform $\Delta A = 284$ ms and $\tau = 0.37$ ms/token, so $\mathrm{ctx}^* \approx 284/0.37 \approx 769$ tokens. At a 100-token context $T_S - T_R = -284 + 0.37 \cdot 100 = -247$ ms (split wins); at a 2,000-token context $T_S - T_R = -284 + 0.37 \cdot 2000 = 456$ ms (replica wins). This is the paper in one line: the second GPU helps as a split for short contexts and as a replica for long ones, with the switch at $\mathrm{ctx}^* \approx \Delta A/\tau$.