# GraniKV: Asymmetric Granularity KV-Cache Paging for Multi-Agent Systems with Long Shared Prefix

**Anonymous ACL submission**


## Abstract

Production paged-serving engines apply uniform paging granularity to the KV cache, even though the two regions of a multi-agent workload have opposite storage requirements: a long shared prefix demands contiguity, while the per-request suffix demands fine-grained allocation.

We present **GraniKV**, a KV-cache layer that allocates the shared prefix in a contiguous HOT pool and the suffix in a token-level COLD pool, combined with a per-step dispatcher which selects the appropriate backend among dual backends for each regime (compute-, memory-, or communication-bound). To the best of our knowledge, GraniKV is the first system to apply asymmetric paging granularity to the KV cache of a production paged-serving engine.

On ShareGPT-V3 user turns prepended with a fixed shared system prompt and served under Poisson arrivals at saturation rate, the gain grows monotonically with the shared-prefix length $L_p$ and is largest at long prefixes — exactly the regime that drives production multi-agent workloads. At $L_p$=16 K shared-prefix tokens GraniKV reaches $\mathbf{2.16}\times$, $\mathbf{1.98}\times$, and $\mathbf{1.57}\times$ the output-token throughput of the production baseline on Llama-3.1-8B/TP=1, Qwen-2.5-14B/TP=2, and Qwen-2.5-32B/TP=4 (TP = tensor-parallel rank count), with no regression at short prefixes.


## 1 Introduction

Multi-agent LLM systems — RAG pipelines, agentic orchestrators, and enterprise assistants with shared system prompts — have become the dominant production deployment pattern for large language models (Liu et al., 2026; Kumar et al., 2025; Lewis et al., 2020). These workloads share one defining feature: a long shared-prefix KV block (system prompt, retrieved document, dialogue history) is identical across many concurrent requests, while each request also carries a unique suffix. Production paged-serving engines — SGLang (Zheng et al., 2024), vLLM (Kwon et al., 2023), and TensorRT-LLM (NVIDIA, 2024) — all apply the same paging granularity to both regions even though they have opposite storage requirements: the shared prefix demands contiguity (so a shared-prefix attention kernel can read it without gather), while the per-request suffix demands fine-grained allocation.

We introduce **GraniKV**, an asymmetric-granularity KV-cache paging layer that splits the KV pool into a contiguous *HOT pool* for shared prefixes and a token-level *COLD pool* for per-request suffix and decoded tokens. To the best of our knowledge, GraniKV is the first system to apply asymmetric paging granularity to the KV cache of a production paged-serving engine, and the first to make fat-GEMM shared-prefix attention (Juravsky et al., 2024) usable inside such an engine. We pair this storage with two attention backends — a dense backend and a cascade backend — and a per-step dispatcher that selects the appropriate backend from the two on every decode step.

On the realistic multi-agent workload — ShareGPT-V3 turns prepended with a fixed system prompt, served under Poisson arrivals at saturation — the win grows monotonically with shared-prefix length $L_p$ and is *largest in long-shared-prefix regimes*, which is precisely where production multi-agent serving operates. At $L_p$=16K, GraniKV reaches $\mathbf{2.16}\times$, $\mathbf{1.98}\times$, and $\mathbf{1.57}\times$ the output-token throughput of the production baseline on Llama-3.1-8B/TP=1, Qwen-2.5-14B/TP=2, and Qwen-2.5-32B/TP=4, with no regression at short prefixes. We expect GraniKV to be most effective on production multi-agent deployments that combine long shared prefixes with datacenter-class hardware; we expect better hardware to push these gains further beyond what we measure on our constrained $4\times$A100 testbed.

## 2 Related Work

### 2.1 Attention, KV Cache, and Paged Allocation

Transformer LLMs (Vaswani et al., 2017) compute multi-head attention at every layer, producing query, key, and value vectors ($Q$, $K$, $V$) per token. Autoregressive decoding must attend over all previous tokens, so the keys and values of every past token are retained throughout generation—forming the *KV cache*—which grows linearly with batch size and context length and dominates HBM consumption in production serving.

Early serving systems allocated a contiguous KV-cache region per request, typically sized to the maximum output length. This caused substantial memory waste from over-reservation and external fragmentation, limiting serving concurrency. Kwon et al. (2023) address this with *PagedAttention*, which partitions the KV cache into fixed-size $P$-token pages allocated on demand and maps logical token indices to physical pages through a per-request block table. PagedAttention resolves external fragmentation while bounding internal fragmentation to at most $P-1$ unused tokens per request. PagedAttention is now the standard KV-cache management mechanism in LLM serving systems.

### 2.2 Multi-Agent Systems and Shared-Prefix Serving

Multi-agent LLM pipelines have become the dominant production deployment pattern (Liu et al., 2026; Kumar et al., 2025). Across these deployments a long *shared prefix* appears in many forms — the retrieved document RAG injects into every batched query (Lewis et al., 2020), the common system prompt enterprise services embed across users, the dialogue history an agentic orchestrator accumulates and replays each turn — and sits identically in front of many concurrent requests.

The shared prefix is already significant in deployed systems — production traces report prefixes reaching 23K tokens with $85-97\%$ of prompt tokens shared across workloads (Srivatsa et al., 2025). As a concrete case study, Anthropic's official Claude Code agent sends a stable shared prefix on every API call — the system prompt, tool definitions, and the project's `CLAUDE.md` instructions — with the conversation history accumulating across turns (Anthropic, 2026a); Anthropic's official prompt-caching design sets the minimum cacheable prefix at 1,024–4,096 tokens depending on the model (Anthropic, 2026b), indicating that production shared prefixes are designed to comfortably exceed this threshold. The engine therefore repeatedly processes the same prefix KV block, while each request carries its own unique suffix.

SGLang (Zheng et al., 2024) addresses prefix sharing with *RadixAttention*: a radix tree keyed on token-id sequences indexes cached KV tensors. Incoming requests traverse the tree to find the longest matching prefix and reuse its KV slots, skipping prefill for those tokens; reference-counted tree nodes keep cached prefixes resident as long as any active request holds them. Recent prefix-aware serving designs — ChunkAttention (Ye et al., 2024a), Mooncake's prefix-cache-aware scheduling (Qin et al., 2025), and hot-prefix replication (Li et al., 2025) — all build on this prefix-reuse idea.

### 2.3 Attention Optimization for Shared Prefixes

Two works exploit the shared-prefix structure by separating prefix from suffix attention. Both methods share the same divide-and-conquer structure—a shared-prefix pass followed by a per-request suffix pass, merged via log-sum-exp —and differ only in which prefix-side inefficiency they target. HydraGen recovers *compute utilization* while cascade inference recovers *memory bandwidth*.

**Shared-prefix matmul (HydraGen).** During decode, each of $B$ requests issues an independent query against the shared prefix, producing $B$ separate matrix-vector products that underutilize Tensor Cores. Juravsky et al. (2024) batch these queries into a single matrix-matrix multiply: if the prefix is stored as a contiguous dense tensor $K_p \in \mathbb{R}^{L_p \times H_{\text{kv}} \times d}$ (Table 1 lists all symbols used in the paper), prefix attention becomes

$$O_p = \text{softmax}\left(\frac{QK_p^\top}{\sqrt{d}}\right) V_p,$$

where $Q$ stacks all $B$ query vectors, exposing the full contraction to Tensor Cores. Per-request suffix attention is computed separately and merged with the prefix output via log-sum-exp.

**Cascade inference.** A complementary bottleneck is redundant HBM traffic: in standard paged decode each of $B$ requests independently reads the full prefix, repeating the same transfer $B$ times per step. Ye et al. (2025) resolve this with a multi-query prefix pass that reads the prefix once per step,

| Sym | Meaning | Sym | Meaning |
|---|---|---|---|
| $L_p$ | prefix len. (tok) | $d$ | per-head dim |
| $L_s$ | suffix len. (tok) | $f_p$ | HOT frac. (0.6) |
| $L_{\text{out}}$ | output len. (tok) | $M_{\text{eff}}$ | $B \cdot n_{\text{q/kv}}$ |
| $B$ | batch size | $(O_p, \ell_p)$ | prefix out, LSE |
| TP | TP rank cnt | $(O_s, \ell_s)$ | suffix out, LSE |
| $H_{\text{kv}}$ | KV heads/rank | | |
| $n_{\text{q/kv}}$ | GQA group size | | |

Table 1: Notation.

a per-request suffix pass, and a log-sum-exp merge:

$$O = \frac{e^{\ell_p - m} O_p + e^{\ell_s - m} O_s}{e^{\ell_p - m} + e^{\ell_s - m}}, \; m = \max(\ell_p, \ell_s), \tag{1}$$

cutting prefix HBM reads from $O(B)$ to $O(1)$.

# 3 GraniKV

Table 1 summarizes the notation used throughout this section.

## 3.1 Design Goals

The methods reviewed in §2 all operate over a KV cache paged with uniform granularity. Cascade inference reduces HBM traffic but still reads the prefix from scattered pages; the shared-prefix matmul achieves Tensor Core saturation only under a contiguous layout that no production paged engine provides. The underlying issue is that both regions of the KV cache are treated identically, even though they have opposite storage requirements—the shared prefix demands contiguity while the per-request suffix demands fine-grained allocation.

GraniKV resolves this by *asymmetric granularity KV-cache paging*, guided by three goals:

1. **Contiguous prefix storage.** The shared prefix KV must occupy a single contiguous memory region for each layer so the attention kernel can load it with a single dense read, with no gather or page-table indirection.
2. **Token-level suffix allocation.** The per-request suffix must be allocated at token granularity to eliminate internal fragmentation.
3. **Regime-adaptive backend dispatch.** The system must pair a compute-optimal and a bandwidth-optimal attention backend with a per-step dispatcher that picks the appropriate one for the live regime.

The first two goals deliver a strict end-to-end improvement over both baselines: GraniKV outperforms the production default (SGLang with FlashInfer) and SGLang with cascade inference once two conditions are satisfied — the shared prefix is long and the in-flight batch saturates the matmul tile. Both conditions are the norm in production multi-agent serving (large system prompts or retrieved contexts, high concurrency to amortize fixed costs), and in that regime GraniKV reaches up to 2.16× the production baseline on Llama-3.1-8B/TP=1 ShareGPT (§4). The third goal guarantees that outside this regime — short prefixes or low concurrency — the dispatch preserves parity for GraniKV rather than causing regressions.

## 3.2 HOT Pool: Contiguous Prefix Storage

The HOT pool is a contiguous bump-pointer allocator over a statically reserved region of GPU HBM. At server boot, a configurable fixed fraction of the total KV budget is carved out for the HOT pool; the remainder is assigned to the COLD pool.

Each HOT allocation services a single shared prefix. When the RadixAttention tree inserts a new prefix node, it requests a contiguous block of $L_p$ token slots from the HOT pool. The allocator advances a bump pointer and returns the base slot index; the prefix KV for layer $l$ and head $h$ is then located at $base + l \times L_p \times H_{\text{kv}} + h \times L_p$—a simple stride computation with no indirection. The tree node records this base index; any attention backend that matches on this prefix receives the base index and prefix length directly.

On eviction — immediately at refcount zero, with no LRU grace window — the range $[base, \; base + L_p)$ is returned to a free list and merged with adjacent free ranges. A subsequent allocation can reuse a freed range without fragmentation. There is no internal fragmentation within an allocation: a prefix either occupies its full contiguous range or is absent.

## 3.3 COLD Pool: Token-Level Suffix Allocator

The COLD pool is a token-level free-list allocator (page size = 1) occupying the KV budget not assigned to the HOT pool. Each generated suffix token receives exactly one slot; there is no intra-page wasted capacity and no partial-page fragmentation. Token-level allocation is appropriate for the suffix because suffix tokens are unique per request: unlike the shared prefix, they are never referenced by more than one request, so larger pages would not enable any cross-request reuse.

**Address-mapping overhead is negligible at decode time.** The per-token page-table indirection cost is empirically small. Ye et al. (2025) report the

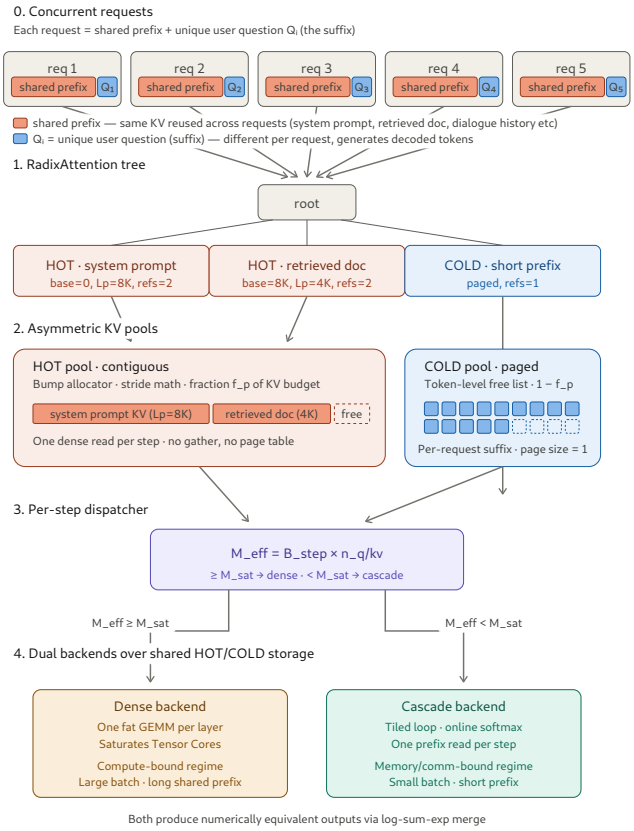


Figure 1: Overview of GraniKV. The RadixAttention tree tags each prefix node as HOT (contiguous in the HOT pool) or COLD (paged fallback); the per-prefix $L_p$ is sized at admission and may differ across prefixes. The per-step dispatcher routes to the dense or cascade backend based on $M_{\text{eff}}$ against a hardware-fitted saturation threshold.

gap between sparse-gather (`page_size`$=1$) and dense-contiguous attention as *within* $1\%$ for decode kernels, and we confirm this at the system level (Appendix B). We adopt page size 1 as the simpler default in order to eliminate internal fragmentation.

### 3.4 Engine Integration

GraniKV extends SGLang (Zheng et al., 2024) — the multi-agent serving engine on which we report all measurements in §4 — with ∼5,000 lines of Python (including the Triton (Tillet et al., 2019) kernels). The host's RadixAttention tree carries one extra piece of metadata per node — a *pool identity*, either a HOT base index or a COLD marker — set at admission and read at decode. Mixed batches are handled in a single kernel; the tree's matching, eviction, and reference-counting logic are unchanged. Figure 1 illustrates the layout of GraniKV.

Any paged-serving system with a prefix-sharing can adopt the same HOT/COLD split by adding a contiguous-allocation primitive and wiring its attention backend to read from either region.

### 3.5 Dual Backends

Production multi-agent deployments span three performance regimes — *compute-bound* (a small model at low TP with a wide batch saturates Tensor Cores), *memory-bound* (a large model at low TP, where KV-cache headroom caps the realizable batch), and *communication-bound* (the same large model at multi-rank TP, where NCCL all-reduce consumes most of decode time and leaves the attention kernel a thin slice). No single kernel is right for all three: compute-bound regimes need fat-contraction Tensor-Core saturation, while memory- and communication-bound regimes leave Tensor Cores idle and need a bandwidth-optimal kernel with a single prefix read per step. GraniKV therefore provides two kernels over the same HOT/COLD storage, one matching each profile.

**The two backends.** Both consume the HOT pool and produce numerically equivalent outputs (Figure 2), but differ in prefix contraction:

- **Dense backend** (inspired by HydraGen). Per

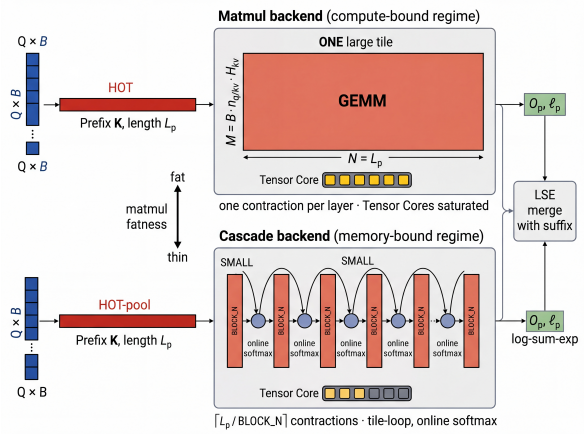


Figure 2: The two attention backends over the shared storage. Both produce numerically equivalent outputs but differ along the matmul-fatness axis (fat GEMMs with no inner loop vs. a tiled loop with online softmax).

layer, one fat batched matmul against the prefix K, a softmax, then one fat batched matmul against the prefix V — covering all requests at once, with no inner loop over the prefix and no online softmax. Suffix runs as paged decode over COLD slots; partial outputs are merged via log-sum-exp (Eq. 1).

- **Cascade backend.** Builds on FlashInfer Cascade (Ye et al., 2024b, 2025), a two-level attention kernel that reads a shared KV region once per step and the per-request suffix separately, merging the two via log-sum-exp; we place the HOT pool as Cascade's level-0 (shared) input. The level-0 kernel remains a FlashAttention-style (Dao et al., 2022; Dao, 2024) tiled loop over BLOCK_N-token chunks with online softmax; the HOT layout only eliminates the `paged_kv_indices` gather of the paged cascade path.

The two differ along one axis—*matmul fatness*. The dense backend issues one $M{=}BQH$, $N{=}L_p$ contraction per layer; cascade issues $\lceil L_p/\text{BLOCK_N} \rceil$ contractions of shape $M{=}Q$, $N{=}\text{BLOCK_N}$. Only the dense backend saturates Tensor Cores at a single contraction; cascade trades peak Tensor Core utilization for tolerance to small effective batches and short prefixes.

Concretely, the dense backend wins in the *compute-bound* regime, cascade wins in the *memory-bound* regime, and the two split the step in the *communication-bound* regime (dense on saturated bursts, cascade on smaller follow-on steps). The per-step dispatch rule that realizes this mapping is given in §3.6, and how each regime is identified empirically is detailed in §4.

Both backends apply only to decode; prefill is delegated unchanged to FlashInfer's prefill kernel, which already reads the HOT pool as a contiguous tensor and the COLD pool through its paged path.

### 3.6 Per-Step Dispatch

The two backends each dominate a regime, but a production serving engine does not see a single steady-state regime. The realized in-flight batch varies step-to-step as requests arrive and complete, and decode steps follow prefill bursts of yet different shapes. A boot-time backend choice would lose on every step that falls on the wrong side of the regime curve.

GraniKV avoids this by dispatching *per step*. A thin wrapper instantiates both children sharing the same HOT/COLD pool, and at each decode step computes the effective matmul $M$-dimension from the realized batch:

$$M_{\text{eff}}(\text{step}) = B_{\text{step}} \times n_{\text{q/kv}}, \qquad (2)$$

where $B_{\text{step}}$ is the step's *realized* in-flight batch — the number of requests actually being decoded on that step, which can exceed the admission batch when prefill bursts from newly arriving requests overlap with ongoing decode — and $n_{\text{q/kv}}$ is the GQA group size. We empirically measured the saturation threshold on A100 used throughout our evaluation and found that Tensor Cores saturate at $M_{\text{eff}} \geq 1024$ (sweep in Appendix B); below this the matmul tile cannot keep all Tensor Core lanes active. The wrapper routes to the dense backend when $M_{\text{eff}} \geq 1024$ and to cascade otherwise. Because $B_{\text{step}}$ is measured each step rather than fixed at admission, sub-threshold cells (e.g. $B{=}128$ at $n_{\text{q/kv}}{=}4$, nominal $M_{\text{eff}}{=}512$) can still route a small fraction of steps to dense whenever the realized batch transiently crosses the threshold; Appendix B (B.4) quantifies this fraction across the synthetic grid. The threshold itself is a hardware property depending on warp count, MMA-instruction shape, and the Tensor Core generation—and would need to be re-fitted on other GPUs. The dispatch *rule* is hardware-agnostic, only the numeric threshold changes.

**Two ceilings, one rule.** Production deployments hit two non-compute-bound ceilings: a *KV-memory ceiling* (weights crowd out KV cache, capping the realized batch) and a *communication ceiling* (all-reduce consumes a growing share of

decode time under TP). The $M_{\text{eff}}$ criterion handles both: memory-bound configurations stay sub-threshold and route every step to cascade, while communication-bound ones cross on saturated bursts (dense) and fall below on smaller steps (cascade) — capturing the dense win where it fires while preserving cascade parity elsewhere. See Appendix B for the per-configuration breakdown.

**Dispatch overhead is negligible.** The wrapper itself adds only sub-microsecond CPU work per decode step ($< 0.2\%$ of per-step wall time at the headline cell); full measurement in Appendix B.

# 4 Experiments

## 4.1 Research Questions

Section 4 answers three questions, each mapped to a subsection below.

**Q1 (§4.5) — Where GraniKV wins.**
Across the synthetic shared-prefix grid, at which $(L_p, B)$ cells and configurations does GraniKV beat the production baseline?

**Q2 (§4.6) — Headline on winning regimes.**
How do the wins look on the three configurations where GraniKV fires, including under realistic Poisson arrivals?

**Q3 (§4.7) — Why GraniKV wins.**
What kernel-level mechanism explains the win?

## 4.2 Setup

**Hardware and software.** We evaluate on a single host with 4×NVIDIA A100-SXM4-40GB (driver 570.195.03, CUDA 12.9, SM clocks locked at 1410 MHz). GPUs are interconnected via NVLink 3.0 (600 GB/s aggregate per GPU); NCCL collectives run over the available NVLink topology. The software stack is PyTorch 2.9.1, Triton 3.5.1, FlashInfer 0.6.8, and NCCL 2.21.

**Models and TP configurations.** We present results on three (model, TP) configurations using Llama-3.1 (Dubey et al., 2024) and Qwen-2.5 (Team, 2024) checkpoints (Table 2): Llama-3.1-8B/TP=1, Qwen-2.5-14B/TP=2, and Qwen-2.5-32B/TP=4. On our 4×A100-SXM4-40GB testbed each TP value hits one of two opposing constraints: low TP leaves too little per-rank HBM for KV cache, so the runtime caps the realized batch below the dense backend's saturation threshold; high TP pays a growing NCCL all-reduce share of decode time, which caps the fraction of the step that shortening attention can recover. We therefore evaluate only the best-fitting TP for each model — the lowest one whose weights fit with enough KV headroom for the target peak concurrency. We report the full TP sweep in Appendix B.

| Model | TP | $n_{\text{q/kv}}$ | KV heads / rank |
|---|---|---|---|
| Llama-3.1-8B | 1 | 4 | 8 |
| Qwen-2.5-14B | 2 | 5 | 4 |
| Qwen-2.5-32B | 4 | 5 | 2 |

Table 2: Evaluation configurations. TP is tensor-parallel rank count; $n_{\text{q/kv}}$ is the GQA query-per-KV-head ratio (see Table 1); "KV heads / rank" is the per-rank KV-head count after TP sharding.

**Decoding.** All systems decode greedily with output tokens capped at 256, so every request deterministically generates the same number of tokens.

## 4.3 Systems Compared

- **SGLang default** — the production default: unmodified SGLang on 16-token paged storage with FlashInfer batched decode.
- **SGLang-Cascade** — FlashInfer Cascade kernel on 16-token paged storage. SGLang-Cascade is exactly GraniKV without the asymmetric-granularity storage layer, providing the ablation of the storage-layer contribution. SGLang upstream has no merged cascade-attention backend, so we implemented the integration ourselves, which is also our standalone contribution.
- **GraniKV** — the per-step adaptive system from §3.6, dispatching each decode step to either the HOT/COLD dense backend or the cascade backend on HOT/COLD storage.

## 4.4 Workload and Metrics

**Synthetic workload.** A controlled grid that strips out arrival-rate dynamics so the regime axes ($L_p$, $B$) can be varied in isolation. Each cell uses a fixed-length shared prefix, a fixed-length unique question, and a synchronous burst of requests — representative of a server operating in steady state at a known concurrency. We sweep $L_p \in \{512, 1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}\}$ tokens crossed with $B \in \{8, 32, 128, 256\}$ concurrent requests; each cell uses a 256-token unique question and injects $3B$ prompts in a synchronous burst.

**Real workload (ShareGPT under Poisson arrivals).** A realistic multi-agent serving trace that combines heterogeneous user turns with stochastic

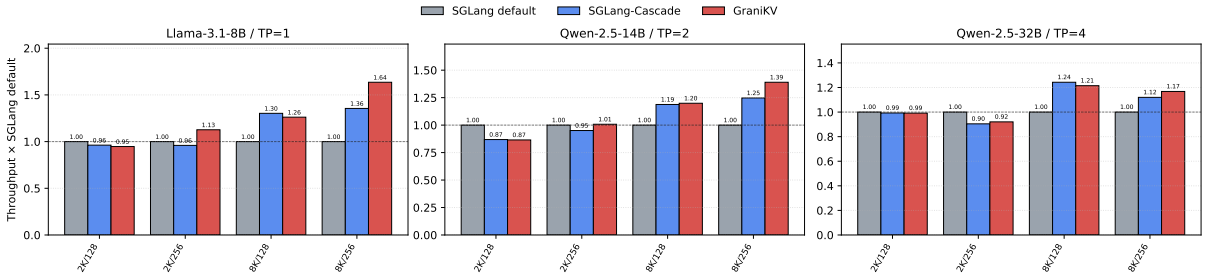


Figure 3: Output-throughput multiplier (vs. SGLang default) on the synthetic shared-prefix grid. Three panels by (model, TP); each cell shows three bars (SGLang default, SGLang-Cascade, GraniKV). The x-axis labels read as $L_p/B$. We show $L_p \in \{2\text{K}, 8\text{K}\}$ crossed with $B \in \{128, 256\}$ (4 cells per panel) — the region where the dense backend can fire. Short-prefix ($L_p$=512) and small-batch ($B$<128) cells, where GraniKV is at parity, are deferred to Appendix B. The dashed line marks 1.0× SGLang-default parity. 3-rep means.

arrival — representative of the production workloads GraniKV is designed for. For each configuration we pair a deterministic $L_p$-token system prompt with 500 ShareGPT-V3 user turns (filtered to $16 \leq |\textit{user turn}| \leq 512$ tokens, deterministic seed 42), generating up to 256 output tokens. We sweep $L_p \in \{512, 1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}\}$ crossed with Poisson arrival rates spanning sub-saturation. The headline cells used in §4.6 are saturation-rate cells from this sweep. To further emulate multi-tenant deployments where distinct shared prefixes share the engine concurrently, we additionally sweep a *mixed-prefix* variant with $k$ distinct $L_p$-token shared prefixes active simultaneously. We sweep $k \in \{1, 4, 16, 64\}$ under the same Poisson regime (full sweep in Appendix B).

**Metrics.** We measure output throughput in tokens per second along with median (p50) latencies for both TTFT (time-to-first-token) and TPOT (time-per-output-token) in milliseconds.

### 4.5 Synthetic Workload Results

Figure 3 reports synthetic-grid throughput across the three configurations; the full per-cell distribution, together with median TTFT and TPOT latencies, appears in Appendix B. GraniKV starts to win where the shared prefix is long and the concurrent batch is large enough to cross the matmul-saturation threshold ($M_{\text{eff}} \geq 1024$ on A100, $B \geq 128$ at our GQA group sizes); at the headline cell ($B$=256, $L_p$=8K), the win reaches 1.64× on Llama-8B/TP=1, 1.39× on Qwen-14B/TP=2, and 1.17× on Qwen-32B/TP=4.

The smaller Qwen-32B/TP=4 win is a testbed constraint: NCCL all-reduce communication overhead consumes a growing share of decode time at 4-way TP, and the realized win would be larger on hosts with higher-bandwidth intra-node GPU interconnects — NVSwitch on H100/H200/B200 hosts or AMD Infinity Fabric on MI300X — than the older NVLink 3.0 (600 GB/s per GPU) on our A100-SXM4 testbed (see Limitations). Outside the dense backend's effective regime (Appendix B), the per-step dispatch guarantees no regression by falling back to cascade.

### 4.6 Real Workload Results

We next check the result under realistic arrival patterns and heterogeneous user turns by sweeping the ShareGPT $L_p$ ladder under Poisson arrivals.

ShareGPT under Poisson arrivals is the operating point that matters most for production multi-agent serving — and the long-shared-prefix regime in particular is where GraniKV's upside is largest. Real multi-agent workloads operate on long shared prefixes such as system prompts, retrieved documents, and accumulated dialogue histories, so the $L_p$ ladder below is the relevant evaluation axis; Table 3 sweeps it at the saturation rate of each cell on the three winning configurations, and the full per-cell distribution appears in Appendix B.

**The wins grow with $L_p$ and are largest at long prefixes.** GraniKV reaches **2.16**× SGLang default at $L_p$=16K on Llama-3.1-8B / TP=1, **1.98**× on Qwen-2.5-14B / TP=2, and **1.57**× on Qwen-2.5-32B / TP=4. On Llama-3.1-8B the win first exceeds 1.5× at $L_p$=4K and roughly doubles by $L_p$=16K; the same monotone-in-$L_p$ scaling holds on both Qwen configurations. These ratios are substantially larger than the synchronous-burst synthetic numbers — the realistic workload amplifies the win because longer prefixes occupy the in-flight batch for longer, and the dense backend's per-step savings accumulate over more decode tokens.

| $L_p$ | SGL | SGL-C | GKV | × SGL | × SGL-C |
|---|---|---|---|---|---|
| *Llama-3.1-8B / TP=1* | | | | | |
| 4K | 3478 | 4209 | 4684 | 1.35× | 1.11× |
| 8K | 2226 | 3476 | 3801 | 1.71× | 1.09× |
| 16K | 1362 | 2800 | 2943 | 2.16× | 1.05× |
| *Qwen-2.5-14B / TP=2* | | | | | |
| 4K | 3702 | 3981 | 4457 | 1.20× | 1.12× |
| 8K | 2489 | 3449 | 3902 | 1.57× | 1.13× |
| 16K | 1567 | 2706 | 3109 | 1.98× | 1.15× |
| *Qwen-2.5-32B / TP=4* | | | | | |
| 4K | 3629 | 3394 | 3597 | 0.99× | 1.06× |
| 8K | 2620 | 3036 | 3292 | 1.26× | 1.08× |
| 16K | 1726 | 2362 | 2707 | 1.57× | 1.15× |

Table 3: ShareGPT throughput at $B$=256 (output tok/s, 3-rep means) on the long-prefix region of the $L_p$ ladder. SGL = SGLang default; SGL-C = SGLang-Cascade; GKV = GraniKV. Full $L_p$ sweep in Appendix B.

**No regression at short prefixes.** At the headline $B$=256, GraniKV is at worst 0.97× SGLang default (Llama-3.1-8B, $L_p$=512) and monotone non-decreasing along the $L_p$ axis. The Poisson workload mixes short and long user turns within each in-flight batch, so the decoder sees requests at different stages on every step; the per-step dispatch routes correctly under this heterogeneity. Regressions at the small-$L_p$ / high-$B$ corners that fall outside the design's effective regime are characterized in Limitations.

**Implication for production multi-agent serving.** The monotone-in-$L_p$ scaling places GraniKV's largest gains exactly in the regime production multi-agent deployments occupy — e.g., Anthropic's Claude Code agent sending its long system prompt, tool definitions, and `CLAUDE.md` on every API call— supporting GraniKV as a strong fit for these workloads.

**Multi-prefix sweep.** The win pattern holds at moderate $k$ on the mixed-prefix variant (§4.4). See the full sweep in Appendix B.

### 4.7 Why GraniKV Wins: a Kernel-Level Microbench

GraniKV's contribution traces to a single mechanism: the contiguous HOT pool turns the dense backend's $QK_p^\top$ contraction from memory-bound into compute-bound. We use a CUDA-event microbench to isolate this by launching the same dense prefix kernel at the headline ($B$=256, $L_p$=8K, Llama-3.1-8B head) against two storage layouts with identical kernel and tile shape: *shared* (HOT) reads each K/V once per step, while *unique* (paged-equivalent) reads the same prefix 256 times.

| Storage | Wall ($\mu$s) | Effective HBM traffic (GB/s) | Speedup |
|---|---|---|---|
| shared (HOT) | 449 | 71 | **15.6**× |
| unique (paged) | 6988 | 1170 | 1× |

Table 4: Kernel-isolated microbench (CUDA-event timing, mean of 20 launches) of the dense prefix kernel at $B$=256, $L_p$=8K on Llama-3.1-8B head shape. "Effective HBM traffic" is total K+V bytes read from HBM divided by wall time. The HOT case's 71 GB/s is low because the kernel reads K/V from HBM only once (subsequent tiles hit L2), not because HBM bandwidth is throttled.

Table 4 reports the headline: storage layout alone moves the dense kernel from HBM-bound to compute-bound, a 15.6× speedup at fixed kernel and problem. The gap between this 15.6× kernel speedup and the measured end-to-end gain is explained by Amdahl's law (Amdahl, 1967):

$$S_{\text{pred}} = \frac{1}{(1-\alpha) + \alpha/s}, \tag{3}$$

where $S_{\text{pred}}$ is the predicted end-to-end speedup over the SGLang-default baseline, $\alpha$ is the fraction of decode wall-time spent in the prefix-attention kernel (measured via torch.profiler on the SGLang-default), and $s$=15.6 is the prefix-attention kernel speedup from Table 4. Across all three configurations the Amdahl prediction over-shoots the measured end-to-end speedup by 14–28%. The residual is attributable to two factors the kernel-only model omits: per-step Python/scheduling overhead that runs at a constant cost per decode step, and NCCL all-reduce traffic that grows with tensor-parallel rank count while attention itself speeds up. The gap therefore tracks TP rank count and is smallest on Llama-8B/TP=1 (+14%, no NCCL contribution). Per-configuration values appear in Appendix B.

## 5 Conclusion

GraniKV applies asymmetric granularity KV cache paging to a production paged-serving engine — a contiguous HOT pool for the shared prefix, a token-level COLD pool for the suffix, and a per-step dispatcher between two attention backends. On the long-shared-prefix, high-concurrency regime of modern multi-agent serving, GraniKV reaches 2.16×, 1.98×, and 1.57× the baseline at $L_p$=16 K across three (model, TP) configurations, with no regression at short prefixes — matching the prefix regime of real multi-agent products. The gains grow as production shared prefixes lengthen.

## Limitations

GraniKV has several limitations that we leave to future work.

**High-tenancy multi-prefix deployments.** GraniKV targets the small-$k$, long-$L_p$ regime exemplified by the production deployments of §2.2 (Claude Code, RAG, long-context QA) — typically one to a handful of distinct system prompts active at a time. In multi-tenant settings with many concurrent distinct prompts (e.g. $k$=64 at $L_p$=8 K = 512 K active tokens; see Appendix B) the active prefix set exceeds the engine's KV budget and all paged engines thrash equally; GraniKV offers no specific benefit in this corner. A hotness-weighted HOT-pool eviction policy (vs. the current strict-refcount-zero release) could narrow the gap when $k$ exceeds the HOT pool's headroom; we leave this to future work.

**Static HOT/COLD pool fraction.** The fraction $f_p$ of the KV budget reserved for the HOT pool is set once at server boot (default $0.6$) and not adjusted. Workloads with shifting prefix-to-suffix ratios across the day — e.g. a daytime mix of long shared system prompts and a nighttime mix dominated by free-form user generation — would benefit from re-sizing $f_p$ online. We leave a dynamic allocator that monitors HOT-pool occupancy and COLD-pool admission pressure and re-partitions the underlying KV buffer between bursts as future work.

**Limited model and hardware coverage.** All measurements come from 5 (model, TP) configurations on a single A100-SXM4-40GB host. The qualitative shape of the per-step dispatch should carry to other architectures, but the numeric saturation threshold ($M \geq 1024$ on A100) is hardware-specific and would need to be re-fitted for H100, MI300X, B200, or future GPUs (§3.6). We also do not evaluate Mixture-of-Experts models, models below 8B, models above 32B, sequence-parallel or expert-parallel deployments, or configurations where the model weights themselves exceed a single rank's HBM (which would foreclose Qwen-32B/TP=1, Llama-70B/TP=4, and similar). Broadening the deployment matrix is straightforward future work but does not change the design.

**Small-$L_p$ / small-$B$ regime.** The asymmetric storage introduces fixed per-step bookkeeping (HOT-pool admission, pool-identity lookup in the radix tree) that does not amortize over short decode runs without compensating prefix savings. In this regime GraniKV is at parity at the $B \leq 256$ short-prefix cells (worst case $0.97\times$ SGLang default on Llama-3.1-8B at $L_p$=512, $B$=256) but degrades further at the small-$L_p$ / high-$B$ corners — up to a $30\%$ regression on Qwen-2.5-14B at $L_p$=512, $B$=512 ($0.70\times$ SGLang default; see Table 9) — where the prefix is too short for the dense backend to saturate Tensor Cores and there is no shared-prefix benefit to amortize the storage layer's per-step costs. Operationally, because GraniKV is activated per server instance via a single `--granikv` CLI flag (§3), deployments that know they will operate in this corner can simply launch without the flag and fall back to the stock paged path; the regression therefore acts as a deployment-time configuration choice rather than a runtime constraint. The mechanism is well-understood, and an admission heuristic that defers HOT promotion until a prefix has accumulated enough references would close the gap inside the GraniKV path itself, but is again left to future work.

## Ethics Statement

GraniKV is an infrastructure-level optimization to the KV-cache layout used by LLM serving engines. It changes how cached attention state is laid out in GPU memory and which kernel computes attention; it does not modify model weights, training data, generation behavior, or any externally visible output of the served model. Bit-exactness against the unmodified baseline serving stack is preserved (§3).

We use publicly released model checkpoints (Llama-3.1-8B, Qwen-2.5-14B, Qwen-2.5-32B) and the publicly available ShareGPT V3 conversation dataset for evaluation; no new data is collected, annotated, or released. The shared-prefix workload generator is synthetic.

The societal impact of more efficient LLM serving is dual-use: lower serving cost makes LLM-backed services more broadly accessible but also lowers the cost of any negative use of those services. The contribution here is at the level of GPU-memory layout, comparable to other inference-optimization work (e.g. PagedAttention, FlashInfer), and inherits the same societal-impact considerations as that body of work.

## Acknowledgements

We thank the reviewers for their feedback, the SGLang and FlashInfer maintainers for the open-source serving stacks on which this work builds, and our colleagues for many helpful discussions and their support during this project. An open Pull Request toward a FlashInfer Cascade attention backend in SGLang upstream was in progress at the time of this submission but had not been merged; our integration was implemented independently, and we plan to submit it as a separate upstream PR. We used AI assistants (large language models) during this work to support literature search, code drafting, and manuscript revision.

## References

Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 483–485.

Anthropic. 2026a. Claude Code: Overview. https://code.claude.com/docs/en/overview. Official documentation for Anthropic's agentic coding tool.

Anthropic. 2026b. Prompt caching. https://platform.claude.com/docs/en/build-with-claude/prompt-caching. Official Claude API documentation; minimum cacheable prefix 1,024–4,096 tokens depending on model.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-throughput LLM inference with shared prefixes. *arXiv preprint arXiv:2402.05099*.

Madabattula Rajesh Kumar, Srinivasa Rao Aravilli, Mustafa Saify, and Shashank Srivastava. 2025. From attention to disaggregation: Tracing the evolution of LLM inference. *arXiv preprint arXiv:2511.07422*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yuhang Li, Rong Gu, Chengying Huan, Zhibin Wang, Renjie Yao, Chen Tian, and Guihai Chen. 2025. HotPrefix: Hotness-aware KV cache scheduling for efficient prefix sharing in LLM inference systems. *Proceedings of the ACM on Management of Data (PACMMOD)*, 3(4):250:1–250:27.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. 2026. Dive into Claude code: The design space of today's and future AI agent systems. *arXiv preprint arXiv:2604.14228*.

NVIDIA. 2024. TensorRT-LLM: A TensorRT toolbox for optimized large language model inference. https://github.com/NVIDIA/TensorRT-LLM.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. In *Proceedings of the 23rd USENIX Conference on File and Storage Technologies (FAST)*.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. 2025. Preble: Efficient distributed prompt scheduling for llm serving. In *International conference on learning representations*, volume 2025, pages 37057–37082.

Qwen Team. 2024. Qwen2.5 technical report. *arXiv preprint*.

Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. 2024a. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Zihao Ye, Lequn Chen, Ruihang Lai, Junru Shao, Yilong Zhao, Size Zheng, Hongyi Jin, and 1 others. 2025. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*.

Zihao Ye, Ruihang Lai, Bo-Ru Lu, Chien-Yu Lin, Size Zheng, Lequn Chen, Tianqi Chen, and Luis Ceze. 2024b. Cascade inference: Memory bandwidth efficient shared prefix batch decoding.

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. 2024. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

# Appendix

## A Source Code and Reproducibility

**Repository.** The GraniKV fork of SGLang is not public during review; it will be disclosed at camera-ready.

## B Full Evaluation Results

**B.1 Full synthetic shared-prefix grid.** Tables 5–7 report the full synthetic-grid sweep for the three winning configurations: $L_p \in \{512, 1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}\}$ crossed with $B \in \{8, 32, 128, 256\}$ (20 cells per configuration). Throughputs are 3-rep means in output tok/s; the rightmost column is GraniKV / SGLang default. At the headline cell ($B$=256, $L_p$=8 K) on Llama-3.1-8B / TP=1, TPOT p50 drops from 109 ms (SGLang default) to 42 ms (GraniKV) — a $\approx$2.6$\times$ per-token speedup. TTFT is comparable across all three systems (within $\sim$ 10% at dense-favourable cells such as $L_p$=8 K, $B$=256; looser at sub-saturation cells where small absolute durations amplify ratios) because prefill is always delegated to FlashInfer's batched prefill.

**Why $L_p$=16 K is omitted from the synthetic grid.** The synthetic burst injector issues each request via a non-streaming call. For prompts up to $L_p$=8 K the per-request response time is short enough that the non-streaming round-trip captures total wall time correctly, and `mean_ttft_ms` grows monotonically with $L_p$ and $B$ as expected. At $L_p$=16 K, however, the captured TTFT collapses to 0 ms across *all* three configurations and *all* batch sizes, and the reported per-cell durations drop to 4–10$\times$ shorter than the $L_p$=8 K row at the same $B$ — physically impossible given the longer prefix. The cause is that the non-streaming injector returns the full completion in one HTTP response, so the bench-side first-token timestamp defaults to 0, and the duration-based throughput calculation incorporates only a fraction of the prefill work. We therefore drop the $L_p$=16 K row from the synthetic grid. *This artifact does not affect any headline number in the paper*: all $L_p$=16 K results in §4.6 (Table 3) and Appendix B B.2 (Tables 8–10) come from the ShareGPT path, which uses `bench_serving` with proper streaming TTFT capture and yields physically consistent $L_p$=16 K numbers (TTFT 1.9–22 s; duration 80–96 s at $B \in \{128, 256, 512\}$).

**B.2 Full ShareGPT $L_p \times B$ matrix.** Tables 8–10 report the full ShareGPT $L_p \times B$ sweep at saturation rate.

**B.2b Concurrency saturation: peak is at $B$=256.** Across all three configurations the GraniKV output throughput is at or near its peak at $B$=256; pushing concurrency to 512 and 1024 yields no further gain and on Qwen-2.5-14B / TP=2 and Qwen-2.5-32B / TP=4 a measurable regression (Table 11). The mechanism is twofold: the per-step dispatcher holds both attention backends simultaneously and pre-allocates CUDA-graph buffers for each, which costs a few hundred MB at $B$=1024 and squeezes the KV-cache budget; and once $M_{\text{eff}} \geq 1024$ is crossed at $B$=256 the dense kernel already saturates Tensor Cores, so additional concurrency adds scheduling and dispatch overhead without improving the kernel's arithmetic intensity. We take $B$=256 as the operating point throughout §4.6–§4.7.

| Config | $B$=256 | $B$=512 | $B$=1024 | Peak $B$ |
|---|---|---|---|---|
| Llama-8B/TP=1 | 2943 | 2960 | 2767 | 512 |
| Qwen-14B/TP=2 | 3109 | 3009 | 3010 | 256 |
| Qwen-32B/TP=4 | 2707 | 2660 | 2685 | 256 |

Table 11: B.2b GraniKV output throughput at $L_p$=16K (tok/s, 3-rep means) across $B \in \{256, 512, 1024\}$. $B$=256 is the peak for two of three configurations and within 0.6% of the Llama peak at $B$=512. Pushing $B$ higher costs throughput on Qwen-14B ($-3\%$) and gives a marginal regression on Qwen-32B/TP=4 ($-1$ to $-2\%$).

**B.3 Cross-Configuration Validity.** The kernel-isolated microbench in §4.7 is by construction kernel-only; we confirm it reproduces in an end-to-end serving setup by capturing torch.profiler

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 540 | 488 | 491 | 0.91× | 14 | 15 | 15 | 0.13 | 0.16 | 0.16 |
| 512 | 32 | 1439 | 1339 | 1314 | 0.91× | 17 | 19 | 19 | 0.58 | 0.56 | 0.61 |
| 512 | 128 | 2846 | 2456 | 2452 | 0.86× | 33 | 40 | 40 | 1.44 | 1.38 | 1.48 |
| 512 | 256 | 3435 | 2892 | 3394 | 0.99× | 51 | 64 | 52 | 2.90 | 2.95 | 2.81 |
| 1K | 8 | 533 | 489 | 484 | 0.91× | 14 | 15 | 15 | 0.16 | 0.16 | 0.18 |
| 1K | 32 | 1353 | 1273 | 1235 | 0.91× | 18 | 19 | 19 | 0.58 | 0.62 | 0.66 |
| 1K | 128 | 2577 | 2321 | 2260 | 0.88× | 35 | 41 | 41 | 1.57 | 1.60 | 1.68 |
| 1K | 256 | 3121 | 2876 | 3257 | 1.04× | 55 | 65 | 52 | 3.00 | 2.87 | 2.97 |
| 2K | 8 | 511 | 477 | 480 | 0.94× | 14 | 15 | 15 | 0.20 | 0.19 | 0.17 |
| 2K | 32 | 1259 | 1310 | 1288 | 1.02× | 21 | 20 | 20 | 0.58 | 0.55 | 0.57 |
| 2K | 128 | 2414 | 2322 | 2287 | 0.95× | 39 | 42 | 41 | 1.70 | 1.63 | 1.78 |
| 2K | 256 | 2816 | 2701 | 3172 | 1.13× | 65 | 68 | 54 | 3.15 | 3.18 | 3.22 |
| 4K | 8 | 484 | 461 | 459 | 0.95× | 15 | 16 | 16 | 0.22 | 0.21 | 0.22 |
| 4K | 32 | 1062 | 1194 | 1226 | 1.15× | 24 | 21 | 21 | 0.68 | 0.66 | 0.59 |
| 4K | 128 | 1970 | 2185 | 2158 | 1.10× | 49 | 43 | 43 | 1.83 | 1.84 | 1.89 |
| 4K | 256 | 2270 | 2568 | 2955 | 1.30× | 83 | 70 | 56 | 3.72 | 3.61 | 3.83 |
| 8K | 8 | 436 | 432 | 434 | 1.00× | 16 | 16 | 16 | 0.26 | 0.27 | 0.25 |
| 8K | 32 | 839 | 1105 | 1126 | 1.34× | 30 | 21 | 22 | 1.03 | 0.96 | 0.83 |
| 8K | 128 | 1469 | 1913 | 1853 | 1.26× | 63 | 41 | 39 | 2.95 | 3.25 | 3.69 |
| 8K | 256 | 1629 | 2208 | **2664** | **1.64×** | 109 | 63 | 42 | 6.03 | 6.62 | 6.64 |

Table 5: B.1 Synthetic grid: Llama-3.1-8B / TP=1. TPOT is median time-per-output-token; TTFT is median time-to-first-token. $L_p$=16 K omitted (see B.1 prose for the measurement artifact).

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 495 | 441 | 445 | 0.90× | 15 | 17 | 16 | 0.20 | 0.20 | 0.20 |
| 512 | 32 | 1402 | 1261 | 1289 | 0.92× | 17 | 19 | 20 | 0.66 | 0.69 | 0.67 |
| 512 | 128 | 2466 | 2011 | 2021 | 0.82× | 38 | 50 | 49 | 1.66 | 1.68 | 1.76 |
| 512 | 256 | 3217 | 2758 | 2828 | 0.88× | 53 | 66 | 63 | 3.24 | 3.22 | 3.34 |
| 1K | 8 | 487 | 442 | 439 | 0.90× | 15 | 16 | 17 | 0.21 | 0.21 | 0.21 |
| 1K | 32 | 1323 | 1222 | 1273 | 0.96× | 18 | 20 | 20 | 0.68 | 0.70 | 0.70 |
| 1K | 128 | 2471 | 1941 | 1999 | 0.81× | 37 | 50 | 48 | 1.74 | 1.73 | 1.82 |
| 1K | 256 | 2946 | 2639 | 2730 | 0.93× | 57 | 68 | 63 | 3.59 | 3.47 | 3.47 |
| 2K | 8 | 474 | 434 | 430 | 0.91× | 15 | 17 | 17 | 0.23 | 0.22 | 0.25 |
| 2K | 32 | 1257 | 1221 | 1232 | 0.98× | 19 | 20 | 20 | 0.76 | 0.76 | 0.76 |
| 2K | 128 | 2263 | 1965 | 1957 | 0.87× | 41 | 49 | 50 | 1.87 | 1.97 | 1.83 |
| 2K | 256 | 2766 | 2629 | 2787 | 1.01× | 64 | 69 | 62 | 3.53 | 3.48 | 3.57 |
| 4K | 8 | 448 | 420 | 418 | 0.93× | 16 | 17 | 17 | 0.27 | 0.27 | 0.28 |
| 4K | 32 | 1116 | 1174 | 1195 | 1.07× | 22 | 20 | 20 | 0.81 | 0.83 | 0.82 |
| 4K | 128 | 1899 | 1862 | 1912 | 1.01× | 49 | 51 | 49 | 2.14 | 2.10 | 2.13 |
| 4K | 256 | 2289 | 2414 | 2617 | 1.14× | 80 | 75 | 67 | 3.79 | 3.81 | 3.74 |
| 8K | 8 | 416 | 398 | 397 | 0.96× | 17 | 18 | 18 | 0.29 | 0.30 | 0.29 |
| 8K | 32 | 893 | 1001 | 1074 | 1.20× | 28 | 22 | 21 | 0.95 | 1.22 | 0.96 |
| 8K | 128 | 1427 | 1695 | 1711 | 1.20× | 65 | 53 | 52 | 2.95 | 2.75 | 2.86 |
| 8K | 256 | 1740 | 2170 | **2419** | **1.39×** | 98 | 71 | 62 | 6.16 | 5.79 | 5.54 |

Table 6: B.1 Synthetic grid: Qwen-2.5-14B / TP=2. TPOT is median time-per-output-token; TTFT is median time-to-first-token. $L_p$=16 K omitted (see B.1 prose for the measurement artifact).

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 411 | 371 | 370 | 0.90× | 18 | 20 | 20 | 0.25 | 0.25 | 0.26 |
| 512 | 32 | 1072 | 985 | 985 | 0.92× | 24 | 26 | 26 | 0.80 | 0.79 | 0.81 |
| 512 | 128 | 2298 | 2112 | 2104 | 0.92× | 39 | 43 | 43 | 2.11 | 2.24 | 2.19 |
| 512 | 256 | 2758 | 2308 | 2326 | 0.84× | 61 | 80 | 78 | 3.96 | 3.94 | 3.99 |
| 1K | 8 | 406 | 367 | 365 | 0.90× | 18 | 20 | 20 | 0.27 | 0.26 | 0.27 |
| 1K | 32 | 1061 | 977 | 976 | 0.92× | 24 | 27 | 26 | 0.80 | 0.80 | 0.80 |
| 1K | 128 | 2253 | 2121 | 2110 | 0.94× | 39 | 43 | 43 | 2.18 | 2.13 | 2.17 |
| 1K | 256 | 2638 | 2250 | 2294 | 0.87× | 65 | 82 | 80 | 4.02 | 3.99 | 3.98 |
| 2K | 8 | 399 | 362 | 362 | 0.91× | 18 | 20 | 20 | 0.28 | 0.28 | 0.29 |
| 2K | 32 | 1023 | 964 | 969 | 0.95× | 24 | 27 | 26 | 0.87 | 0.85 | 0.85 |
| 2K | 128 | 2087 | 2072 | 2069 | 0.99× | 43 | 44 | 44 | 2.23 | 2.22 | 2.22 |
| 2K | 256 | 2499 | 2259 | 2300 | 0.92× | 69 | 80 | 77 | 4.18 | 4.16 | 4.28 |
| 4K | 8 | 389 | 355 | 355 | 0.91× | 18 | 20 | 20 | 0.33 | 0.31 | 0.31 |
| 4K | 32 | 968 | 939 | 927 | 0.96× | 26 | 27 | 27 | 0.93 | 0.90 | 0.94 |
| 4K | 128 | 1831 | 1973 | 1973 | 1.08× | 51 | 47 | 46 | 2.37 | 2.31 | 2.36 |
| 4K | 256 | 2190 | 2104 | 2190 | 1.00× | 82 | 86 | 81 | 4.36 | 4.40 | 4.44 |
| 8K | 8 | 367 | 344 | 345 | 0.94× | 19 | 21 | 21 | 0.34 | 0.33 | 0.31 |
| 8K | 32 | 857 | 884 | 875 | 1.02× | 29 | 28 | 28 | 1.05 | 1.01 | 1.12 |
| 8K | 128 | 1459 | 1812 | 1772 | 1.21× | 66 | 48 | 50 | 2.74 | 2.79 | 2.77 |
| 8K | 256 | 1767 | 1979 | **2064** | **1.17×** | 101 | 85 | 79 | 5.60 | 5.63 | 5.74 |

Table 7: B.1 Synthetic grid: Qwen-2.5-32B / TP=4. TPOT is median time-per-output-token; TTFT is median time-to-first-token. $L_p$=16 K omitted (see B.1 prose for the measurement artifact).

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 586 | 532 | 531 | $0.91\times$ | 13 | 15 | 15 | 0.1 | 0.1 | 0.1 |
| 512 | 32 | 1882 | 1696 | 1692 | $0.90\times$ | 16 | 18 | 18 | 0.2 | 0.2 | 0.3 |
| 512 | 128 | 4556 | 3666 | 3602 | $0.79\times$ | 25 | 32 | 32 | 0.7 | 0.8 | 0.7 |
| 512 | 256 | 6284 | 4859 | 5151 | $0.82\times$ | 34 | 46 | 42 | 1.4 | 1.3 | 1.4 |
| 512 | 512 | 5928 | 4796 | 5915 | $1.00\times$ | 63 | 82 | 63 | 2.1 | 2.2 | 2.2 |
| 512 | 1024 | 5954 | 4765 | 4563 | $0.77\times$ | 63 | 82 | 89 | 2.1 | 2.1 | 2.2 |
| 1K | 8 | 577 | 527 | 526 | $0.91\times$ | 13 | 15 | 15 | 0.1 | 0.1 | 0.1 |
| 1K | 32 | 1789 | 1679 | 1680 | $0.94\times$ | 17 | 18 | 18 | 0.2 | 0.3 | 0.2 |
| 1K | 128 | 4233 | 3576 | 3581 | $0.85\times$ | 27 | 32 | 33 | 0.7 | 0.8 | 0.8 |
| 1K | 256 | 5559 | 4716 | 5087 | $0.92\times$ | 39 | 47 | 40 | 1.6 | 1.4 | 1.7 |
| 1K | 512 | 5206 | 4511 | 5859 | $1.13\times$ | 70 | 84 | 63 | 2.3 | 2.3 | 2.4 |
| 1K | 1024 | 5222 | 4540 | 4472 | $0.86\times$ | 70 | 85 | 90 | 2.3 | 2.7 | 2.3 |
| 2K | 8 | 563 | 520 | 520 | $0.92\times$ | 14 | 15 | 15 | 0.1 | 0.1 | 0.1 |
| 2K | 32 | 1624 | 1639 | 1641 | $1.01\times$ | 18 | 18 | 18 | 0.3 | 0.3 | 0.3 |
| 2K | 128 | 3669 | 3488 | 3502 | $0.95\times$ | 31 | 33 | 33 | 0.8 | 0.8 | 0.8 |
| 2K | 256 | 4741 | 4606 | 5030 | $1.06\times$ | 46 | 48 | 40 | 1.8 | 1.6 | 1.6 |
| 2K | 512 | 4295 | 4357 | 5758 | $1.34\times$ | 84 | 85 | 62 | 3.1 | 3.1 | 3.1 |
| 2K | 1024 | 4421 | 4113 | 4394 | $0.99\times$ | 83 | 85 | 88 | 3.2 | 3.1 | 3.1 |
| 4K | 8 | 533 | 500 | 500 | $0.94\times$ | 14 | 15 | 15 | 0.1 | 0.1 | 0.2 |
| 4K | 32 | 1385 | 1581 | 1569 | $1.13\times$ | 22 | 19 | 19 | 0.3 | 0.4 | 0.4 |
| 4K | 128 | 2823 | 3309 | 3278 | $1.16\times$ | 41 | 34 | 34 | 1.2 | 1.0 | 1.1 |
| 4K | 256 | 3478 | 4209 | 4684 | $1.35\times$ | 63 | 50 | 38 | 2.5 | 2.6 | 2.5 |
| 4K | 512 | 3311 | 3920 | 5359 | $1.62\times$ | 111 | 86 | 58 | 4.9 | 4.8 | 4.9 |
| 4K | 1024 | 3258 | 3932 | 4117 | $1.26\times$ | 111 | 87 | 87 | 4.9 | 5.1 | 5.2 |
| 8K | 8 | 488 | 483 | 484 | $0.99\times$ | 16 | 16 | 16 | 0.2 | 0.2 | 0.2 |
| 8K | 32 | 1054 | 1466 | 1470 | $1.39\times$ | 28 | 19 | 19 | 0.4 | 0.6 | 0.6 |
| 8K | 128 | 1940 | 2791 | 2752 | $1.42\times$ | 58 | 37 | 38 | 2.2 | 2.2 | 2.1 |
| 8K | 256 | 2226 | 3476 | 3801 | $1.71\times$ | 97 | 55 | 47 | 5.0 | 5.1 | 5.5 |
| 8K | 512 | 2110 | 3091 | 3485 | $1.65\times$ | 173 | 98 | 96 | 9.4 | 9.5 | 9.8 |
| 8K | 1024 | 2101 | 3076 | 3379 | $1.61\times$ | 172 | 104 | 95 | 9.6 | 9.3 | 9.7 |
| 16K | 8 | 416 | 446 | 449 | $1.08\times$ | 18 | 17 | 17 | 0.2 | 0.3 | 0.3 |
| 16K | 32 | 722 | 1194 | 1278 | $1.77\times$ | 42 | 24 | 23 | 0.3 | 0.3 | 0.3 |
| 16K | 128 | 1231 | 2349 | 2355 | $1.91\times$ | 98 | 47 | 47 | 0.6 | 0.8 | 0.5 |
| 16K | 256 | 1362 | 2800 | 2943 | $2.16\times$ | 160 | 65 | 59 | 10.8 | 10.1 | 10.5 |
| 16K | 512 | 1339 | 2671 | 2960 | $2.21\times$ | 262 | 85 | 73 | 20.2 | 20.0 | 20.2 |
| 16K | 1024 | 1331 | 2663 | 2767 | $2.08\times$ | 262 | 86 | 71 | 20.3 | 19.9 | 20.3 |

Table 8: B.2 ShareGPT $L_p \times B$ at saturation rate: Llama-3.1-8B / TP=1. TPOT is median time-per-output-token; TTFT is median time-to-first-token.

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 538 | 480 | 479 | $0.89\times$ | 14 | 16 | 16 | 0.1 | 0.1 | 0.1 |
| 512 | 32 | 1842 | 1591 | 1621 | $0.88\times$ | 16 | 19 | 19 | 0.3 | 0.3 | 0.3 |
| 512 | 128 | 3725 | 2870 | 2881 | $0.77\times$ | 31 | 41 | 41 | 0.9 | 0.8 | 0.8 |
| 512 | 256 | 5797 | 4390 | 4710 | $0.81\times$ | 37 | 50 | 47 | 1.6 | 1.6 | 1.6 |
| 512 | 512 | 6581 | 5042 | 4603 | $0.70\times$ | 64 | 87 | 97 | 2.6 | 2.6 | 2.7 |
| 512 | 1024 | 6494 | 5025 | 4581 | $0.71\times$ | 64 | 88 | 96 | 2.7 | 2.6 | 2.8 |
| 1K | 8 | 533 | 478 | 479 | $0.90\times$ | 15 | 16 | 16 | 0.1 | 0.1 | 0.1 |
| 1K | 32 | 1758 | 1613 | 1609 | $0.91\times$ | 17 | 19 | 19 | 0.3 | 0.3 | 0.3 |
| 1K | 128 | 3738 | 2803 | 2795 | $0.75\times$ | 30 | 42 | 42 | 0.9 | 0.9 | 0.9 |
| 1K | 256 | 5465 | 4288 | 4654 | $0.85\times$ | 39 | 52 | 47 | 1.7 | 1.6 | 1.7 |
| 1K | 512 | 6084 | 4953 | 4556 | $0.75\times$ | 70 | 88 | 97 | 2.7 | 2.8 | 2.8 |
| 1K | 1024 | 6066 | 5009 | 4493 | $0.74\times$ | 70 | 87 | 97 | 2.7 | 2.7 | 3.0 |
| 2K | 8 | 514 | 474 | 471 | $0.92\times$ | 15 | 16 | 16 | 0.1 | 0.1 | 0.1 |
| 2K | 32 | 1626 | 1585 | 1586 | $0.98\times$ | 18 | 19 | 19 | 0.3 | 0.3 | 0.3 |
| 2K | 128 | 3532 | 2828 | 2817 | $0.80\times$ | 32 | 41 | 42 | 1.0 | 0.9 | 0.9 |
| 2K | 256 | 4719 | 4244 | 4567 | $0.97\times$ | 46 | 52 | 48 | 1.9 | 1.8 | 1.8 |
| 2K | 512 | 5218 | 4777 | 4835 | $0.93\times$ | 83 | 91 | 90 | 3.1 | 3.2 | 3.1 |
| 2K | 1024 | 5191 | 4770 | 4881 | $0.94\times$ | 83 | 91 | 89 | 3.2 | 3.1 | 3.0 |
| 4K | 8 | 498 | 465 | 463 | $0.93\times$ | 15 | 17 | 17 | 0.1 | 0.1 | 0.1 |
| 4K | 32 | 1405 | 1531 | 1522 | $1.08\times$ | 21 | 19 | 20 | 0.3 | 0.3 | 0.4 |
| 4K | 128 | 2862 | 2718 | 2682 | $0.94\times$ | 40 | 43 | 44 | 1.1 | 1.1 | 1.1 |
| 4K | 256 | 3702 | 3981 | 4457 | $1.20\times$ | 58 | 53 | 47 | 2.5 | 2.6 | 2.5 |
| 4K | 512 | 4033 | 4467 | 4292 | $1.06\times$ | 105 | 91 | 96 | 4.8 | 5.0 | 5.0 |
| 4K | 1024 | 4012 | 4433 | 4281 | $1.07\times$ | 104 | 93 | 96 | 4.9 | 5.0 | 5.2 |
| 8K | 8 | 465 | 442 | 443 | $0.95\times$ | 16 | 17 | 17 | 0.2 | 0.2 | 0.2 |
| 8K | 32 | 1107 | 1419 | 1422 | $1.28\times$ | 27 | 20 | 20 | 0.6 | 0.5 | 0.6 |
| 8K | 128 | 2017 | 2450 | 2403 | $1.19\times$ | 56 | 45 | 47 | 1.8 | 1.7 | 1.5 |
| 8K | 256 | 2489 | 3449 | 3902 | $1.57\times$ | 85 | 57 | 48 | 4.3 | 4.3 | 4.4 |
| 8K | 512 | 2607 | 3652 | 3755 | $1.44\times$ | 155 | 99 | 97 | 8.9 | 9.1 | 8.6 |
| 8K | 1024 | 2649 | 3676 | 3789 | $1.43\times$ | 153 | 100 | 96 | 8.7 | 8.7 | 8.8 |
| 16K | 8 | 410 | 418 | 416 | $1.01\times$ | 18 | 18 | 18 | 0.3 | 0.3 | 0.3 |
| 16K | 32 | 783 | 1251 | 1254 | $1.60\times$ | 39 | 23 | 23 | 0.5 | 0.6 | 0.7 |
| 16K | 128 | 1320 | 2137 | 2141 | $1.62\times$ | 91 | 53 | 53 | 0.7 | 0.8 | 0.9 |
| 16K | 256 | 1567 | 2706 | 3109 | $1.98\times$ | 138 | 69 | 59 | 8.7 | 8.8 | 9.1 |
| 16K | 512 | 1635 | 2866 | 3009 | $1.84\times$ | 229 | 97 | 93 | 17.5 | 17.8 | 17.1 |
| 16K | 1024 | 1621 | 2870 | 3010 | $1.86\times$ | 232 | 98 | 92 | 17.7 | 17.6 | 17.3 |

Table 9: B.2 ShareGPT $L_p \times B$ at saturation rate: Qwen-2.5-14B / TP=2. TPOT is median time-per-output-token; TTFT is median time-to-first-token.

| | | Output (tok/s) | | | | TPOT (ms) | | | TTFT (s) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $L_p$ | $B$ | SGL | SGL-C | GKV | $\times$ SGL | SGL | SGL-C | GKV | SGL | SGL-C | GKV |
| 512 | 8 | 445 | 399 | 397 | $0.89\times$ | 17 | 19 | 20 | 0.1 | 0.1 | 0.1 |
| 512 | 32 | 1320 | 1186 | 1181 | $0.89\times$ | 23 | 26 | 26 | 0.4 | 0.4 | 0.4 |
| 512 | 128 | 3718 | 3309 | 3261 | $0.88\times$ | 30 | 34 | 35 | 1.1 | 1.1 | 1.1 |
| 512 | 256 | 4813 | 3646 | 3703 | $0.77\times$ | 44 | 61 | 60 | 2.0 | 1.9 | 2.0 |
| 512 | 512 | 5883 | 4615 | 3929 | $0.67\times$ | 71 | 94 | 112 | 3.2 | 3.3 | 3.3 |
| 512 | 1024 | 5825 | 4608 | 3942 | $0.68\times$ | 71 | 94 | 112 | 3.2 | 3.3 | 3.3 |
| 1K | 8 | 443 | 398 | 398 | $0.90\times$ | 17 | 20 | 20 | 0.1 | 0.1 | 0.1 |
| 1K | 32 | 1297 | 1177 | 1180 | $0.91\times$ | 23 | 26 | 26 | 0.4 | 0.4 | 0.4 |
| 1K | 128 | 3550 | 3237 | 3240 | $0.91\times$ | 32 | 35 | 35 | 1.0 | 1.1 | 1.0 |
| 1K | 256 | 4823 | 3656 | 3746 | $0.78\times$ | 44 | 61 | 59 | 2.0 | 2.0 | 1.9 |
| 1K | 512 | 5540 | 4494 | 3891 | $0.70\times$ | 76 | 96 | 114 | 3.3 | 3.3 | 3.4 |
| 1K | 1024 | 5541 | 4556 | 3965 | $0.72\times$ | 75 | 95 | 111 | 3.3 | 3.2 | 3.3 |
| 2K | 8 | 436 | 394 | 395 | $0.90\times$ | 18 | 20 | 20 | 0.2 | 0.1 | 0.2 |
| 2K | 32 | 1256 | 1164 | 1171 | $0.93\times$ | 24 | 26 | 26 | 0.4 | 0.4 | 0.4 |
| 2K | 128 | 3255 | 3229 | 3195 | $0.98\times$ | 35 | 35 | 35 | 1.2 | 1.2 | 1.1 |
| 2K | 256 | 4410 | 3422 | 3623 | $0.82\times$ | 49 | 67 | 61 | 2.0 | 2.1 | 2.0 |
| 2K | 512 | 5024 | 4455 | 4275 | $0.85\times$ | 84 | 97 | 101 | 3.4 | 3.4 | 3.6 |
| 2K | 1024 | 4994 | 4417 | 4356 | $0.87\times$ | 84 | 98 | 99 | 3.5 | 3.4 | 3.5 |
| 4K | 8 | 429 | 390 | 390 | $0.91\times$ | 18 | 20 | 20 | 0.2 | 0.2 | 0.2 |
| 4K | 32 | 1174 | 1149 | 1148 | $0.98\times$ | 25 | 26 | 26 | 0.4 | 0.5 | 0.4 |
| 4K | 128 | 2732 | 3045 | 3015 | $1.10\times$ | 42 | 37 | 38 | 1.3 | 1.3 | 1.1 |
| 4K | 256 | 3629 | 3394 | 3597 | $0.99\times$ | 60 | 64 | 60 | 2.6 | 2.6 | 2.6 |
| 4K | 512 | 4147 | 4222 | 3765 | $0.91\times$ | 100 | 99 | 113 | 5.1 | 4.9 | 5.0 |
| 4K | 1024 | 4149 | 4208 | 3741 | $0.90\times$ | 100 | 99 | 114 | 5.0 | 5.0 | 5.0 |
| 8K | 8 | 407 | 380 | 379 | $0.93\times$ | 19 | 20 | 20 | 0.2 | 0.2 | 0.2 |
| 8K | 32 | 1050 | 1095 | 1085 | $1.03\times$ | 28 | 27 | 27 | 0.6 | 0.6 | 0.6 |
| 8K | 128 | 2061 | 2724 | 2678 | $1.30\times$ | 56 | 41 | 41 | 1.6 | 1.5 | 1.6 |
| 8K | 256 | 2620 | 3036 | 3292 | $1.26\times$ | 78 | 66 | 59 | 4.5 | 4.5 | 4.6 |
| 8K | 512 | 3015 | 3466 | 3294 | $1.09\times$ | 130 | 106 | 113 | 8.6 | 9.1 | 9.2 |
| 8K | 1024 | 3007 | 3608 | 3336 | $1.11\times$ | 130 | 102 | 113 | 8.7 | 8.7 | 8.9 |
| 16K | 8 | 374 | 358 | 357 | $0.95\times$ | 20 | 21 | 21 | 0.3 | 0.3 | 0.3 |
| 16K | 32 | 877 | 1009 | 1001 | $1.14\times$ | 34 | 28 | 28 | 0.9 | 1.0 | 1.0 |
| 16K | 128 | 1382 | 2159 | 2162 | $1.56\times$ | 86 | 52 | 51 | 1.1 | 1.4 | 1.5 |
| 16K | 256 | 1726 | 2362 | 2707 | $1.57\times$ | 121 | 79 | 69 | 9.4 | 9.0 | 8.9 |
| 16K | 512 | 1938 | 2815 | 2660 | $1.37\times$ | 185 | 103 | 114 | 17.1 | 17.1 | 17.5 |
| 16K | 1024 | 1934 | 2774 | 2685 | $1.39\times$ | 184 | 104 | 113 | 17.5 | 17.6 | 17.4 |

Table 10: B.2 ShareGPT $L_p \times B$ at saturation rate: Qwen-2.5-32B / TP=4. TPOT is median time-per-output-token; TTFT is median time-to-first-token.

traces at the $B$=256, $L_p$=8K cell on two configurations — the Llama-3.1-8B / TP=1 headline (Table 12) and a structurally different Qwen-2.5-32B / TP=4 (Table 13, $M_{\text{eff}}$=1280) — for all four systems. SGLang-Cascade reduces prefix-attention kernel time by 3.66× on Llama-3.1-8B / TP=1 (11058 → 3021 ms) and by 1.24× on Qwen-2.5-32B / TP=4 (20298 → 16423 ms); the smaller reduction at the larger configuration is consistent with its lower $H_{\text{kv}}/\text{rank}$=2, which leaves fewer heads for the multi-query cascade kernel to deduplicate across. SGL-C/GKV (SGLang-Cascade run on GraniKV's HOT/COLD storage) tracks SGLang-Cascade within trace-noise on every category at both configurations — within 1.5% at Qwen-32B/TP=4 and within 5% at Llama-3.1-8B/TP=1 (max 4.5% on the Linear GEMM category, where short per-step kernels make ratios noisier) — confirming the storage layer is kernel-agnostic. GraniKV's dense-backend contribution does not surface in either CUPTI capture because its Triton-compiled kernel is invisible to the in-server profiler hook; `nsys` (Table 16) confirms the kernel fires by direct CUDA-event capture.

| Category | SGL | SGL-C | SGL-C/GKV | GKV |
|---|---|---|---|---|
| Prefix attention | 11058 | 3021 | 3005 | 3040 |
| Decode attention | 95 | 286 | 279 | 281 |
| Linear GEMM | 9558 | 10011 | 9563 | 9766 |
| Total kernel | 21365 | 14088 | 13585 | 13855 |

Table 12: Kernel-time category breakdown at Llama-3.1-8B / TP=1, $B$=256, $L_p$=8K (ms, summed over the trace window). Columns as in Table 13.

| Category | SGL | SGL-C | SGL-C/GKV | GKV |
|---|---|---|---|---|
| Prefix attention | 20298 | 16423 | 16671 | 16345 |
| Decode attention | 612 | 1066 | 1076 | 1069 |
| Linear GEMM | 29396 | 29618 | 29791 | 29784 |
| Total kernel | 127914 | 136525 | 138822 | 139422 |

Table 13: Kernel-time category breakdown at Qwen-2.5-32B / TP=4, $B$=256, $L_p$=8K (ms, summed over the trace window). SGL = SGLang default; SGL-C = SGLang-Cascade; SGL-C/GKV = SGLang-Cascade kernel run on GraniKV's HOT/COLD storage.

**B.4 Per-Step Dispatch Trace.** Table 14 reports the fraction of decode steps routed to the dense backend on Llama-3.1-8B / TP=1 across the synthetic-grid $(L_p, B)$ cells. Cells crossing $M_{\text{eff}}=B \cdot n_{\text{q/kv}} \geq 1024$ route almost entirely to matmul, sub-threshold cells to cascade — empirical evidence that routing follows the realized batch, not a static boot-time choice.

| | Matmul-routed fraction at $B$ | | | | |
|---|---|---|---|---|---|
| $L_p$ | 8 | 32 | 128 | 256 | majority |
| 512 | 0% | 0% | 4% | 98% | cascade |
| 2K | 0% | 0% | 7% | 99% | cascade |
| 8K | 0% | 0% | 11% | 100% | mixed |
| 16K | 0% | 0% | 13% | 100% | mixed |

Table 14: B.4 Fraction of decode steps the per-step wrapper routes to the dense backend, on Llama-3.1-8B / TP=1 ($n_{\text{q/kv}}$=4). At $B$=256 the saturation threshold $M_{\text{eff}} \geq 1024$ is met on every step; at $B$<256 the realized concurrency under prefill bursts occasionally crosses the threshold and matmul fires opportunistically.

**B.4b Empirical fit of the dispatch threshold.** The $M_{\text{eff}}$≥1024 threshold of §3.6 was selected by examining the per-token kernel cost of the dispatcher's two children at fixed $L_p$=8 K on Llama-3.1-8B / TP=1 ($n_{\text{q/kv}}$=4; Table 15, drawn from the same synthetic-grid runs as Table 5). Below $M_{\text{eff}}$=1024 the dense backend's per-token TPOT is within 3% of cascade and the dispatcher routes ≤13% of steps to dense (Table 14); at $M_{\text{eff}}$=1024 the dense backend's per-token TPOT drops to 0.67× cascade and the dispatcher routes 98−100% of steps to dense. The knee is sharp and sits at $M_{\text{eff}}$∼1024 on A100, justifying the threshold; on other hardware it would be re-fit by re-running this sweep.

| $B$ | $M_{\text{eff}}$ | GKV TPOT (ms) | SGL-C TPOT (ms) | GKV / SGL-C |
|---|---|---|---|---|
| 8 | 32 | 16 | 16 | 1.00× |
| 32 | 128 | 22 | 21 | 1.02× |
| 128 | 512 | 39 | 41 | 0.97× |
| 256 | 1024 | 42 | 63 | **0.67×** |

Table 15: B.4b Empirical fit of the dispatch threshold: Llama-3.1-8B / TP=1, $L_p$=8 K synthetic-grid cells. $M_{\text{eff}}=B \cdot n_{\text{q/kv}}$ with $n_{\text{q/kv}}$=4. The crossover from cascade-parity to a clear dense win occurs between $M_{\text{eff}}$=512 and $M_{\text{eff}}$=1024, justifying the ≥ 1024 dispatcher cutoff.

**B.5 Per-Step Dispatch Overhead.** The thin Python wrapper that selects the kernel each step is not on the critical path. An `nsys` capture under `--trace=cuda,nvtx,osrt` on the dense-backend microbench over a 15-second window measures the values in Table 16: the dense-backend prefix kernel fires 83 times at 442 $\mu$s per launch (CUPTI direct capture), and the Python wrapper adds < 1 $\mu$s of CPU work per decode step — under

0.2% of the per-step wall time at the headline cell. The same `nsys` trace serves as direct evidence that the dense backend actually fires under per-step dispatch (the kernel is invisible to the in-server torch.profiler hook — see Table 13).

| Quantity | Value |
|---|---|
| Dense-backend launches in 15 s | 83 |
| Kernel wall / launch | 442 $\mu$s |
| Wrapper CPU / step | $< 1\,\mu$s |
| Wrapper % of step | $< 0.2\%$ |

Table 16: Per-step dispatch overhead from `nsys` on the dense-backend microbench.

**B.6 COLD Page-Size Sensitivity.** We verify that token-level COLD allocation (page size 1) does not regress throughput relative to coarser paging by sweeping the COLD page size $\in \{1, 4, 16\}$ on the headline cells of two configurations, holding HOT pool and attention kernels fixed (Table 17). End-to-end throughput is flat within $\pm 0.2\%$ across the sweep — well inside our per-cell 3-rep noise floor. The per-token indirection overhead introduced by page size 1 does not materialize in end-to-end measurement.

| Config | PS=1 | PS=4 | PS=16 | $\lvert\Delta\rvert_{\max}$ |
|---|---|---|---|---|
| Llama-8B / TP=1 | 2664 | 2661 | 2659 | 0.19% |
| Qwen-32B / TP=4 | 2064 | 2065 | 2062 | 0.15% |

Table 17: B.6 COLD page-size sensitivity at the headline cell ($B$=256, $L_p$=8K) of two configurations. Throughput in output tok/s (3-rep mean); $\lvert\Delta\rvert_{\max}$ is the largest spread within each row. All three page sizes track within $\pm 0.2\%$.

**B.7 TP Regime Envelope.** GraniKV's benefit depends on two structural ceilings that move in opposite directions with tensor-parallel rank count. The *KV-memory ceiling* shrinks as TP shrinks: at low TP, weights consume more per-rank HBM, leaving less room for KV cache, so the runtime caps the realized batch below the matmul-saturation threshold. The *NCCL Amdahl ceiling* shrinks as TP grows: the all-reduce share of decode time grows with rank count, capping the fraction of the step that shortening attention can recover. A related *head-fused-tile floor*, $H_{\text{kv}}/\text{rank} = H_{\text{kv}}/\text{TP}$, collapses at $H_{\text{kv}}/\text{rank}{=}1$ and prevents the dense backend from saturating Tensor Cores at all. Table 18 summarizes the empirical pattern across five (model, TP) configurations, all measured at the synthetic-grid headline cell ($L_p$=8 K, $B$=256). The three configurations that also appear in B.1 (Llama-8B/TP=1, Qwen-14B/TP=2, Qwen-32B/TP=4) match the corresponding B.1 8K/256 rows exactly; Qwen-14B/TP=4 and Qwen-32B/TP=2 are additional cells run at the same point to mark the NCCL-Amdahl and KV-memory ceiling boundaries, respectively, and are not part of the headline B.1 sweep.

| Config | $H_{\text{kv}}$/rk | NCCL | KV mem | $\frac{\text{GKV}}{\text{SGL}}$ | Regime |
|---|---|---|---|---|---|
| Llama-8B / TP=1 | 8 | none | abund. | 1.64× | peak |
| Qwen-14B / TP=2 | 4 | 2-way | comf. | 1.39× | modest NCCL |
| Qwen-14B / TP=4 | 2 | 4-way | comf. | 1.18× | NCCL-bound |
| Qwen-32B / TP=4 | 2 | 4-way | tight | 1.17× | NCCL+mem |
| Qwen-32B / TP=2 | 4 | 2-way | v.tight | 0.99× | KV-ceil |

Table 18: TP regime envelope across five (model, TP) configurations at the synthetic-grid headline cell ($L_p$=8 K, $B$=256). The Qwen-32B/TP=2 row is the parity case that bounds GraniKV's contribution from below.

**Why TP=8 is not evaluated.** At TP=8 on a 4× or 8×A100 host, every model in our matrix lands at $H_{\text{kv}}/\text{rank}{=}1$, putting it on the wrong side of the head-fused-tile floor regardless of model size. The per-step dispatch correctly routes every step to cascade, so a TP=8 experiment would only measure the cascade-integration sub-contribution (already characterized in §4.5–§4.6) at a more communication-heavy operating point. There is no recommended single TP: the operator should pick the lowest TP that fits the model's weights with sufficient KV headroom for the target peak concurrency, and GraniKV's per-step dispatch then maximizes the available kernel choice within that envelope.

**B.8 Multi-prefix sensitivity.** The ShareGPT setup of §4.6 prepends a single fixed system prompt to every request — the most favourable case for the HOT pool. To more deeply simulate realistic production workloads, we also evaluate GraniKV under a *multi-shared-prefix* setting that mirrors a multi-tenant deployment where several distinct shared prefixes share the engine concurrently. We sweep the prefix-set cardinality $k \in \{1, 4, 16, 64\}$ on Llama-3.1-8B / TP=1 at the $L_p$=8 K, $B$=256 headline cell. For each $k$, we generate $k$ lexically-distinct synthetic system prompts of length $L_p$ with deterministic seeds; every incoming user turn is deterministically assigned one of the $k$ prompts. Because the system prompts here are synthetically generated rather than the ShareGPT-derived fixed prompt of §4.6, the $k$=1 row is the within-

experiment single-prefix baseline for this variant and is numerically close to but not identical to the $L_p$=8 K, $B$=256 row of Table 8 (within ∼4% across all three systems); the discrepancy reflects differing token-level prompt content at fixed $L_p$, not a methodological change.

| System | $k$=1 | $k$=4 | $k$=16 | $k$=64 |
|---|---|---|---|---|
| SGLang default | 2300 | 2239 | 1334 | 304 |
| SGLang-Cascade | 3602 | 2201 | 1314 | 295 |
| GraniKV | 3881 | 3034 | 1417 | 296 |
| GraniKV / SGL | 1.69× | 1.36× | 1.06× | 0.97× |
| GraniKV / SGL-C | 1.08× | 1.38× | 1.08× | 1.00× |

Table 19: B.8 Multi-prefix sensitivity at Llama-3.1-8B / TP=1, $L_p$=8 K, $B$=256. Output tok/s, 3-rep means. $k$ is the cardinality of the distinct-system-prompt set; each request is deterministically assigned one of the $k$ prompts. The $k$=1 row recovers the single-prefix headline.

**Findings.** GraniKV retains a solid win at $k$=4 (1.36× over SGLang default, 1.38× over SGLang-Cascade — in fact a *larger* cascade margin than $k$=1, because 4-way prefix split reduces in-batch cascade sharing while the HOT pool still holds all four prefixes simultaneously). At $k$=16 all three systems converge to near-parity as $16 \times L_p = 128$K tokens compete for the HOT/KV budget; at $k$=64 all three collapse together (≈ 300 tok/s, an 87% drop from $k$=1) because $64 \times L_p = 512$K tokens exceed the realizable working set under $B$=256 concurrency. The collapse at $k$=64 is a paged-engine-wide phenomenon, not a GraniKV-specific limitation: every system thrashes when the active prefix set exceeds the engine's KV budget.

**B.9 Amdahl residual breakdown.** The Amdahl prediction in §4.7 over-predicts the measured end-to-end gain by 14–28%, with the gap monotonically tracking TP rank count (Table 20). Two factors not captured by the kernel-only model account for the residual: (i) fixed per-step overhead — Python dispatch, scheduling, prefix admission — that does not scale with kernel speed; and (ii) at TP>1, NCCL all-reduce time that grows with rank count but stays constant under the kernel speedup. Llama-8B/TP=1 has the smallest gap (+14%) because there is no NCCL contribution; the larger gaps on Qwen-14B/TP=2 (+28%) and Qwen-32B/TP=4 (+25%) match the TP regime envelope of Appendix B.

| Config | $\alpha$ | Amdahl | Measured | Gap |
|---|---|---|---|---|
| Llama-8B/TP=1 | 63.5% | 2.47× | 2.16× | +14% |
| Qwen-14B/TP=2 | 64.6% | 2.53× | 1.98× | +28% |
| Qwen-32B/TP=4 | 52.6% | 1.97× | 1.57× | +25% |

Table 20: Amdahl decomposition at the headline ShareGPT cell ($L_p$=16 K, $B$=256). $\alpha$ is the prefill-attention fraction (torch.profiler, SGLang default). Prediction over the measured gain by 14–28%.

**B.10 Cross-pool admission and promotion policy.** At admission, GraniKV's allocator tries the HOT pool (§3.2) first. If the HOT pool lacks contiguous capacity for the requested $L_p$ — either because the pool is full or because free fragments are too small — the prefix is placed in the COLD pool, with no demotion of any existing HOT prefix. The no-demotion rule keeps the dense backend's stride-math addressing into the HOT pool statically valid: a request mid-decode that holds a HOT base index can continue without re-checking residency on every step. A prefix that lands in COLD stays in COLD for its lifetime — we do not promote on subsequent reference, because the per-decode-step KV copy required to move the prefix's K/V tensors between pools exceeds the per-step kernel advantage the move would unlock on the workloads we measure. Hotness-aware admission heuristics (e.g. deferring HOT promotion until a prefix has accumulated enough references) could refine this policy and are noted as future work in the Limitations section.