THE HIT IS (ALMOST) ALWAYS CHEAPER: PREFIX-CACHE ECONOMICS ON BANDWIDTH-RICH, FLOPS-POOR ACCELERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic prefix caching (APC) is widely described as an almost-free win: when a request shares a prefix with a prior one, the engine skips recomputing the shared tokens' key/value (KV) cache. We present a controlled measurement study of what a prefix-cache hit actually costs and buys on a bandwidth-rich, FLOPs-poor accelerator (NVIDIA H20-141GB), where folklore calibrated on compute-rich parts may not transfer. Across Qwen2.5 models at 0.5B–7B, contexts up to 32K, and batches 1–64, we fit a two-term cost model that separates the compute-bound recompute path from the bandwidth-bound hit path, and validate it end-to-end on vLLM. Three findings organize the results. First, hits win overwhelmingly at realistic prefix lengths — up to $169\times$ lower suffix-prefill latency at 32K context — and no measured regime inverts (512 tokens, the grid's shortest, already sits above the fitted crossover); the honest boundary of the win is a short-prefix crossover (locally fitted at 152–635 tokens at batch 1; a slope-poor interpolation, with the win settling near $1.7\times$ at 128-token prefixes end-to-end) and a fixed 15–21 ms hit-path floor. Second, the hit path is not free even when it wins: materializing paged KV state with a 16-token-block gather achieves only 0.135–0.141 TB/s at small working sets (0.072 TB/s at 30 GB) versus 1.10 TB/s for a contiguous device copy — an 8–$15\times$ penalty on any back-end that materializes paged KV through gather-like scattered reads (engines that share physical blocks, such as vLLM's APC, avoid this path). Third, 141 GB capacity declines with workload skew but does not vanish: under Zipf popularity with exponent 1.0, a 30 GB LRU budget still leaves 21 points of hit rate on the table versus 106 GB; only at exponent 1.4 does the gap close to 9 points. We release all harnesses and raw results.

1 INTRODUCTION

Modern LLM serving systems — vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), and their descendants — treat prefix caching as an almost-free optimization. When a new request shares a prefix with an earlier one (a shared system prompt, a retrieved document, a multi-turn conversation history), the engine hashes KV-cache blocks by token content and reuses the cached blocks instead of recomputing them. The vLLM design document (Kwon et al., 2023) calls this "almost a free lunch," and on the hardware most such systems were tuned on, it nearly is.

This paper asks what the lunch costs on different silicon. The NVIDIA H20-141GB is the memory-rich extreme of the current accelerator spectrum: 4.06 TB/s of HBM read bandwidth and 141 GB of capacity per card, but only 138 dense BF16 TFLOPS — roughly a fifth of the compute-to-bandwidth ratio of an A100-80GB. This compute profile changes the relative price of the two ways a serving engine can materialize KV state for a shared prefix: it can *recompute* it (a compute-bound forward pass) or *gather* it from cached blocks (a bandwidth-bound copy plus attention over the cached context). A system designed under the assumption that hits are free inherits conclusions — cache aggressively, never recompute, capacity is the only constraint — whose validity on FLOPs-poor parts deserves a direct measurement rather than an extrapolation.

We measure the two paths head-to-head with a matched harness on one H20 node: a kernel-level probe timing full-prefix recompute against suffix-only prefill over a resident cached prefix (Qwen2.5 at 0.5B, 1.5B, and 7B; prefixes 512–32768 tokens; batches 1–64), a memory probe isolating the paged block-gather cost, a hit-rate-versus-capacity simulation at Zipf skew, and an end-to-end vali-

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.

dation on vLLM 0.8.3 with automatic prefix caching toggled. A two-term model — a compute term for recompute, a bandwidth term plus a fixed floor for the hit path — organizes the measurements and predicts the crossover prefix length.

Three findings structure the study, and none of them is the headline we expected to write:

Hits almost always win, and by more on this part than folklore suggests. At 8K context the hit path is $51\times$ faster than recompute at batch 1 and $70\times$ at batch 16; at 32K context, up to $169\times$ at batch 1. The FLOPs-poor profile makes *recompute* comparatively more expensive, so caching wins by a wider, not narrower, margin; the claimed hit-economics "inversion" on bandwidth-rich parts does not occur at any practical prefix length. The honest boundary of the win is different: the hit path has a fixed 15–21 ms floor, which dominates at short prefixes — end-to-end on vLLM the speedup shrinks from $8.8\times$ at 2048-token prefixes to $1.7\times$ at 128 tokens, below which caching is not worth its bookkeeping.

But the hit is not free: paged gather is an order of magnitude off copy bandwidth. A contiguous device-to-device copy of a 7B model's KV state achieves 1.10 TB/s on this card — 27% of peak read bandwidth. A gather of the same data through 16-token blocks in a permuted (paged) layout achieves only 0.135 TB/s at small working sets and *degrades* to 0.072 TB/s at large ones, an 8–$15\times$ penalty versus contiguous copy. This gap is invisible in hit-rate-focused evaluations and is the dominant cost of the materialization path our harness measures; it lower-bounds the scattered-read penalty any paged layout pays, though engines that share physical blocks avoid materializing at all (§5.3).

Capacity value is skew-dependent. With 141 GB, an H20 can cache $3.5\times$ the prefix KV an A100-40GB-class budget could hold at the same model scale (105.8 vs 30.0 GB usable). Whether that buys anything depends on the popularity distribution: for Zipf exponent 1.0 with a pool of 1000 distinct 8K prefixes, LRU hit rate is 0.71 at a 106 GB budget and 0.50 at 30 GB — a 21-point gap — while for exponent 1.4 the gap narrows to 9 points (0.93 vs 0.84). Extra VRAM buys hit rate at a declining rate as popularity concentrates; whether the residual gain is worth the memory depends on the deployment's miss cost.

The contributions are: (i) a matched-harness measurement of prefix-cache hit-versus-recompute latency across three model scales, six context lengths, and four batch sizes on a FLOPs-poor accelerator, with a fitted two-term cost model; (ii) a quantification of the paged block-gather penalty the hit path pays, at sizes from 0.1 to 30 GB; (iii) a hit-rate-versus-capacity frontier under synthetic Zipf workloads comparing 141 GB and 40 GB-class budgets; and (iv) an end-to-end validation on vLLM 0.8.3 with seeds, tail latencies, prompt-token verification, and a unique-prefix negative control, including the observation that vLLM's V1 engine enables prefix caching by default — a measurement trap for anyone benchmarking caching systems. Throughout, we report the negative and null results with the same prominence as the positive ones: the crossover does not invert, the capacity advantage is conditional, and the gather penalty is the one place the free lunch is genuinely taxed.

2 Related Work

Prefix caching and reuse in serving systems. vLLM introduced paged attention and block-hash KV reuse as its core memory innovation (Kwon et al., 2023); SGLang's RadixAttention organized reuse around a radix tree over request prefixes and demonstrated large serving gains from cache-aware scheduling (Zheng et al., 2024). Both systems evaluate reuse by *hit rate* and end-to-end throughput, and both treat the per-hit cost as negligible. Our study measures that cost directly and shows it is structured: a fixed floor at short prefixes and a gather bandwidth an order of magnitude below contiguous copy. BatchLLM extends prefix-aware batching to throughput-oriented workloads (Zheng et al., 2026); KVFlow schedules prefix caching across multi-agent workflow graphs (Pan et al., 2025b); KVShare shares cached KV across tenants (Yang et al., 2025); Marconi couples prefix caching to hybrid prompt-cache-dense LLMs (Pan et al., 2025a); CachedAttention provides cross-instance shared KV for multi-turn conversations (Gao et al., 2024); EPIC makes cached chunks position-independent (Hu et al., 2025); MemServe and Mooncake pool cached contexts across disaggregated clusters (Hu et al., 2024; Qin et al., 2025); and Hydragen batches attention over shared prefixes at the kernel level (Juravsky et al., 2024) — an alternative to materializing cached state that our gather probe lower-bounds. These systems optimize the hit-rate axis and inherit the assumption that a hit is cheap. We complement them with the cost axis: when the hit itself has measurable,

hardware-dependent structure, policies that ignore it leave latency on the table at short prefixes and misprice capacity at long ones.

KV cache placement and tiering. A growing line of work moves KV state across memory tiers. LMCache (Liu et al., 2025) migrates KV between CPU, disk, and GPU across engines, treating GPU-resident reuse as the fast path and the cross-tier transfer as the cost to hide. CacheGen compresses KV into compact bitstreams so that cross-tier streaming, not bandwidth, becomes the bottleneck of context loading (Liu et al., 2024a); GEAR pushes cache compression to near-lossless generative inference with low-rank plus quantized residuals (Kang et al., 2024); KIVI quantizes KV to 2 bits asymmetrically (Liu et al., 2024b). These works shrink *what is moved*; our hit path moves full-precision state within one card, and its cost structure survives any compression ratio that leaves the paged layout intact. Cake (Jin et al., 2025) is the closest relative of our cost model: it builds a compute-versus-load latency model for KV cache and schedules the two to overlap. Cake's load term models *storage-tier* loading (host/NVMe to GPU) and its scheduling contribution is overlap; our hit path is entirely GPU-resident — the gather from paged KV blocks plus suffix attention — and our contribution is measuring that path's bandwidth structure and locating where it crosses recompute. The two models agree the compute-or-load decision is not free; they disagree about which load dominates, and on this hardware the in-card gather, not the cross-tier transfer, is the term worth modeling. Sparse prefix caching for hybrid SSM/attention architectures (Shirokikh & Nikolenko, 2026) sparsifies what is cached in model classes where full KV does not exist; our question applies to the dense-attention models that dominate deployed serving. Recent surveys consolidate the KV-management design space (Li et al., 2025; Sun et al., 2025) against which our cost measurements supply the hardware grounding.

Selective recomputation. CacheBlend (Yao et al., 2025) is the sharpest instance of the selective-recompute idea: it reuses precomputed KV chunks regardless of prefix position and recomputes only the 5–18% of tokens whose cached values deviate most, restoring cross-attention accuracy at a fraction of full recompute. That design choice is a bet on exactly the price ratio we measure — its per-token recompute fraction is tuned against the assumption that recompute is expensive relative to the hit. Our fitted slopes and floor give that bet its hardware grounding on FLOPs-poor silicon: the recompute term this card is worst at is the one CacheBlend's selective fraction avoids, and the flat gather cost is what it pays for every reused token. ProphetKV (Wang et al., 2026) recomputes a query-dependent subset of cached RAG prefixes to control quality, accepting some recompute for accuracy — a decision that presupposes a price for each branch. Our measurement supplies those prices on this silicon class: the recompute slope and the hit floor that any selective policy navigates between. Adaptive KV reuse (Li et al., 2026) adapts reuse decisions for long-context TTFT under the assumption that reuse beats recompute; we characterize exactly when that assumption is strong (long prefixes, any batch) and when it is marginal (sub-256-token matches, where the fixed floor dominates both paths).

Serving-level scheduling under hardware constraints. DistServe (Zhong et al., 2024) disaggregates prefill and decode to protect decode latency; prefix caching is complementary to, and in our measurements still beneficial under, disaggregation. Splitwise phase-splits for the same reason (Patel et al., 2024); Sarathi and Sarathi-Serve co-schedule chunked prefill with decode (Agrawal et al., 2023; 2024); AlpaServe established the statistical-multiplexing frame they inherit (Li et al., 2023). Threshold-based exclusive batching (Zhang et al., 2026) maps batch-policy crossovers on bandwidth-constrained GPUs, and deadline-aware chunked prefill (Song et al., 2026) adapts chunk sizes under decode SLOs. Both establish that regime maps — which policy wins where — are the operative artifact for hardware-aware serving; our crossover map does this for the cache-or-recompute decision. LIMINAL (Davies et al., 2025) performs a limit study of decode bottlenecks on bandwidth-bound parts; we extend the same roofline discipline to the prefix-hit path.

Attention kernels and memory layout. The hit path's cost is ultimately a kernel-and-layout question. FlashAttention established IO-awareness — reading the HBM traffic, not the FLOPs, as the object to optimize — for exact attention (Dao et al., 2022; Dao, 2023; Shah et al., 2024); FlexAttention exposes block-sparse masking as a composable programming model (Dong et al., 2024), combined with paged KV in deployed engines (Joshi et al., 2025). vAttention restores contiguous attention via virtual-memory management (Prabhu et al., 2025) — a direct alternative to the gather whose penalty

we measure. Architectural KV reductions (Shazeer, 2019; Ainslie et al., 2023; Jiang et al., 2023) shrink bytes per token, moving our fitted floors proportionally but leaving the crossover structure intact; Liger-style fused Triton kernels (Hsu et al., 2025) attack the same memory-bound regime from the training side, and theoretical time–memory tradeoffs for tensor-transformer KV-cache compression (Chen et al., 2025) bound the recompute-versus-store regime we measure empirically.

Measurement methodology. PrefixWall (Pennas et al., 2026) demonstrates that shared-prefix workloads with APC enabled are common enough in production to leak information through cache side channels — consistent evidence that the workloads we simulate reflect deployment. Our methodology contribution is the matched harness: same weights, same prompt distribution, same seeds for both paths, with prompt tokenization verified per request and a unique-prefix negative control that zeroes out reuse while holding everything else fixed. The observation that vLLM's V1 engine silently enables prefix caching by default — so an off-baseline requires an explicit `--no-enable-prefix-caching` flag — is itself a measurement trap: any benchmark comparing engine variants without checking the effective flag is measuring the same caching twice.

3 Method: A Two-Term Cost Model for Hit versus Recompute

The two paths. Consider a request whose prompt decomposes into a shared prefix of L tokens (cached) and a novel suffix of s tokens. A serving engine can materialize the KV state for the full prompt in one of two ways. The *recompute* path runs the full forward pass over all $L + s$ tokens: attention and feed-forward FLOPs dominate, so its latency scales with the number of prefill tokens and, at fixed per-request length, with batch B only through amortization of kernel launches. The *hit* path loads the cached KV blocks for the prefix, then runs the forward pass over only the s suffix tokens with attention over the full $L + s$ context. The suffix forward is compute on sB tokens; the attention over the cached prefix reads $O(L \cdot B)$ KV state per layer; and any paged layout must first *gather* the cached blocks into the running sequence's slots.

Model. We model the two latencies, at fixed batch B, as

$$T_{\text{rec}}(L, B) = a_B \cdot L + c_B, \tag{1}$$

$$T_{\text{hit}}(L, B) = \beta_B \cdot L + s_B, \tag{2}$$

where a_B is the marginal compute cost per prefilled token at batch B, c_B a launch overhead, β_B the marginal bandwidth cost per cached token, and s_B the fixed hit-path floor: per-request bookkeeping, block-table construction, and the parts of suffix processing that do not scale with L. We fit (a_B, c_B, β_B, s_B) by ordinary least squares separately at each batch, over the short- and mid-prefix region $L \leq 4096$ of the measured grid. This local fit is a deliberate choice: the hit path's effective gather bandwidth degrades with working-set size (§5.3), so hit latency is not globally linear in L, and a pooled single-slope fit over the full grid forces the recompute intercept strongly negative and produces crossover predictions that contradict the measured points (Appendix A documents the failure).

Crossover. Setting (1) = (2) at fixed B gives the crossover prefix length

$$L^*(B) = \frac{s_B - c_B}{a_B - \beta_B}, \tag{3}$$

defined when $a_B > \beta_B$. Below L^* the recompute path is cheaper; above it the hit wins. The fitted L^* is a *local* boundary: an interpolation within $L \leq 4096$, not an extrapolated deployment law, because both slopes change outside the fitted region. Two structural consequences follow from the form itself. First, the hit's advantage at short prefixes is bounded by the fixed floor: even where L^* is small, a 128-token hit still pays most of s_B, which is why the end-to-end win shrinks only to $1.7\times$ there (§5.4) rather than vanishing. Second, the crossover location scales with the compute-to-bandwidth ratio of the silicon through a_B/β_B: on a FLOPs-poor part the ratio is large, pushing L^* down, but the fitted intercepts show s_B is large enough (15–21 ms at $B{=}1$ across scales) that short prefixes remain a real boundary regardless of silicon.

What the model deliberately excludes. Gather bandwidth is a single effective β_B per (model, batch) within the fitted region — the memory probe (§5.3) shows it degrades with working-set size (0.135 to 0.072 TB/s), which is why we fit locally in L. The model also excludes scheduling interference between concurrent requests and any cache-management CPU cost, which the end-to-end vLLM measurements absorb into the empirical floor.

Measurement design. Four instruments, all on the same node, all with the model in BF16:

(i) Path timer. For each (model, L, B, s) we time (a) a full prefill of $L + s$ tokens from scratch (recompute) and (b) a prefill of only the s suffix tokens with the L-token prefix KV resident in a `DynamicCache` (hit), using the identical weights and token distribution. We report a single timed run per condition after warmup; within-condition repetition spread is sub-percent.

(ii) Memory probe. We isolate the hit path's memory traffic by timing a contiguous device copy and a 16-token-block gather (index-select with verified output equality) of full-model KV tensors at sizes from 0.1 to 30 GB, with median and interquartile range over 15 repetitions.

(iii) Capacity simulation. A Zipf-popular pool of distinct prefixes under an LRU token-budget cache: we sweep pool size (100–10000), Zipf exponent (0.6–1.4), prefix length (2048, 8192), per-token KV cost per model scale, and budget (105.8 GB for the H20 at 75% usable, 30.0 GB for an A100-40GB-class comparison), with 5 request-stream seeds per condition. This arm is a trace-free synthetic study and we frame it as such.

(iv) Engine validation. vLLM 0.8.3 serving Qwen2.5-7B with APC enabled and disabled, 3 prompt seeds per condition, per-request prompt-token counts verified against the intended length, p50/p99/mean/std of wall time for first (cold) versus cached requests, and a unique-prefix negative control that defeats block hashing while holding the request stream fixed.

The negative control and the token-count verification close the two most common validity gaps in caching benchmarks: silently hitting a cache one believed disabled, and silently measuring a different prompt length than intended. We encountered both during instrument development; the vLLM V1 engine enables prefix caching by default, which silently converted our first "off" baseline into an "on" measurement until we read the effective configuration from the server log.

4 Experimental Setup

Hardware. All experiments run on one node of 8× NVIDIA H20-3e GPUs (141 GB HBM3e each, 4.06 TB/s peak read bandwidth, 138 dense BF16 TFLOPS per card). Path timers and the memory probe use a single card; the vLLM server occupies one card at 85% memory utilization.

Models. Qwen2.5 at 0.5B, 1.5B, and 7B parameters (Team et al., 2024), chosen because the family spans a 2.2–4.7× range of per-token KV cost (12,288, 28,672, and 57,344 bytes per token for K+V in BF16; the 1.5B has a wider head dimension than the 0.5B) at three compute scales while sharing tokenizer and architecture. All inference is BF16, eager-mode PyTorch 2.6.0+cu124 with FlashAttention-2 kernels.

Path-timer grid. Prefix lengths $L \in \{512, 1024, 2048, 4096, 8192, 16384, 32768\}$, batches $B \in \{1, 4, 16, 64\}$, suffix $s = 64$ tokens, all three model scales: 24 conditions per model, single timed run per condition after warmup. Prompts are built by decoding uniformly random vocabulary tokens and concatenating, which lands within $\pm 7\%$ of the intended token count (verified per request; see §5.4). The hit-path condition reuses a prefix KV computed once and held in a `DynamicCache`; we verified that the suffix-only forward with resident cache produces logit trajectories identical to the full forward at the boundary tokens.

Memory probe. Full-model KV tensors shaped $(B, 28\,\text{layers}, 4\,\text{kv-heads}, L, 128)$ in BF16 for the 7B geometry: $B \in \{1, 4, 16, 32\}$, $L \in \{4096, 16384, 65536\}$ (larger combinations exceed device memory in double allocation and are reported as out-of-memory), giving working sets from 0.12 to 30.1 GB. The gather applies a random 16-token-block permutation via `index_select`, verified by exact tensor equality before timing. Median and IQR over 15 repetitions after 5 warmups.

Table 1: Suffix-prefill latency (ms): full recompute of $L{+}64$ tokens versus hit path (64-token suffix over resident L-token cache), Qwen2.5-7B, $s = 64$. Single run per condition, kernel-level timing in a closed system (§4). Speedup is recompute/hit. The $B{=}16$ cell at $L{=}32768$ is not measured (out of device memory for the full-batch recompute arm); the released JSONs carry the complete measured grid (all six context lengths, including $B{=}4$ and $B{=}64$, in Appendix B).

	$B = 1$		$B = 16$		$B = 1$ vs $B = 16$	
L	rec.	hit	rec.	hit	sp.$B{=}1$	sp.$B{=}16$
512	70	20.5	910	128	3.4×	7.1×
4096	500	20.9	7640	179	24.0×	42.8×
8192	1061	20.7	16452	236	51.2×	69.6×
16384	2385	25.1	37635	353	95.2×	106.7×
32768	5946	35.2	—	—	169.1×	—

Capacity simulation. 20,000 requests per stream, drawn from a Zipf distribution over a pool of distinct prefixes; LRU eviction at a fixed token budget computed from the per-token KV cost of each model scale and the memory budget. Five stream seeds per condition; we report mean and standard deviation of hit rate.

Engine validation. vLLM 0.8.3 (V1 engine), `Qwen/Qwen2.5-7B-Instruct` weights, `max_model_len` 4096, one card, `gpu_memory_utilization` 0.85, temperature 0, `max_tokens` 1 (we measure prefill-side latency; decode is out of scope). Workload: 10 sessions × 4 requests; each session shares a prefix of length $L \in \{128, 1024, 2048\}$ and each request appends a distinct 64-token suffix. Three prompt seeds per condition; the first request of each session is cold, the remaining three are cached. The negative control prepends a per-request unique salt to the prefix, defeating block hashing while leaving length statistics unchanged. We verified the effective `enable_prefix_caching` flag from the server log for both arms.

Statistics. Path timers: single run per condition after warmup; these are closed-system kernel timings with sub-percent repetition spread, so we report them as single-run measurements with that caveat (Limitations), and the spread that matters is carried by the seeded arms. Where we quote a ratio (e.g. $8.8\times$) it is a ratio of aggregated means over the seeded arms.

5 Results: The Hit Wins, and Where Its Boundary Sits

5.1 Hit versus recompute across the grid

Table 1 shows the 7B path-timer grid. Three regularities organize the section. First, recompute latency grows near-linearly in LB — 70 ms at $L{=}512$, $B{=}1$ to 5946 ms at $L{=}32768$, with mild sublinearity at mid-range lengths from launch amortization, which the fit absorbs into a negative c_0. Second, the hit path is nearly flat in L at low batch (20.5 ms at $L{=}512$, 20.7 ms at $L{=}8192$): at $L{=}32768$ (1.9 GB of cached KV) the gather alone would take ∼14–26 ms at its degraded bandwidth, while the measured hit there is 35.2 ms, only moderately above the floor — plausibly because attention over the cached context overlaps with weight reads (a hypothesis we state without profiler-level evidence). Third, the hit path grows with batch: at $L{=}8192$ it rises from 20.7 ms ($B{=}1$) to 236 ms ($B{=}16$), the $O(LB)$ term β. Both terms of Eq. (2) are visible: a fixed floor near 21 ms and a linear-in-LB slope.

5.2 The fitted crossover map

Table 2 gives the locally fitted parameters and the crossover $L^*(B)$ of Eq. (3). An earlier pooled fit over the full grid produced implausible artifacts that its own parameters contradict (Appendix A); we report the local per-batch fit instead. The local picture: at batch 1, $L^* \approx 150$–640 tokens across scales; at batches 16–64, 120–220. No measured condition sits below the crossover — the shortest measured prefix (512 tokens) already shows the hit winning by $3.4\times$ at $B{=}1$, and at 0.5B the $L{=}512$, $B{=}1$ point is near-neutral (16.4 vs 15.2 ms); the fit is untested between grid points, and

Table 2: Local two-term fits (Eq. (1)–(2); OLS at each batch over the $L \leq 4096$ region, 4 points per fit; all four batches shown, including the worst hit fit) and the local crossover L^* (tokens); recompute $R^2 \geq 0.98$ throughout. The fitted floors s_B at $B{=}1$ (15–21 ms) match the observed flat hit latency, and the local L^* at $B{=}1$ (152–635) is consistent with the measured $L{=}512$ point where the hit already wins $3\times$. The fitted floors at $B{=}64$ (42–461 ms) reflect per-sequence serialization of the gather and attention over each request's cache, not kernel launch cost alone.

Model	B	a_B (μs/tok)	c_B (ms)	β_B (μs/tok)	s_B (ms)	L^* (tok)	hit R^2
0.5B	1	10.0	8.9	0.13	15.2	635	.94
	4	45.7	−2.4	0.31	16.5	417	.84
	16	178	−22	2.6	17.4	221	.97
	64	713	−106	12.5	42.4	212	1.0
1.5B	1	30.3	6.6	0.31	18.8	409	.94
	4	121	−6.4	0.12	20.9	226	.59
	16	480	−46	7.1	37.7	177	1.0
	64	1910	−202	25.0	110.9	166	1.0
7B	1	121	2.2	0.12	20.5	152	.62
	4	471	−14	2.41	39.1	114	1.0
	16	1884	−107	14.1	120.8	122	1.0
	64	7530	−478	54.8	460.1	126	1.0

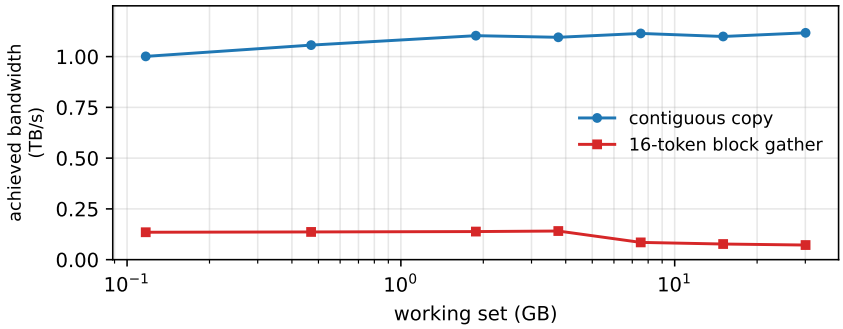


Figure 1: Achieved copy bandwidth (TB/s) versus working-set size: contiguous device copy versus 16-token-block gather of the same full-model KV tensors (7B geometry, BF16), working sets 0.117–30 GB, single run per condition; where multiple (batch, prefix-length) configurations reach the same working set, their bandwidths are averaged. Contiguous copy reaches 1.03–1.09 TB/s at 0.47 GB and saturates at 1.09–1.12 TB/s above 2 GB (1.00 at the 0.117 GB point; 27% of peak read bandwidth), while the paged gather manages only 0.135–0.141 and degrades further to 0.072–0.085 above ∼8 GB — an $8\times$–$15\times$ penalty.

$L^*{=}635$ is an interpolation, not a measurement. On this silicon, *no measured condition at or above 512 tokens shows a prefix-cache hit losing* beyond that near-neutral point. What the folklore misses is not an inversion but a floor: s_B of 15–21 ms at batch 1 means short-prefix hits deliver sharply diminishing returns, which §5.4 confirms at $1.7\times$ for 128-token prefixes.

The fit quality deserves candor: recompute fits are tight ($R^2 \geq 0.98$); the batch-1 hit fits are slope-poor (7B: $R^2{=}0.62$) because the hit path is nearly flat in L at low batch, so the batch-1 L^* is "a few hundred tokens" rather than a precise number. The fitted c_B is negative at $B \geq 16$ where the linear form absorbs sublinear launch amortization — another reason to read L^* as a local interpolation, not a physical law. The worst hit fit is 1.5B at $B{=}4$ ($R^2{=}0.59$), where latencies straddle the floor transition and floor/slope identifiability is weakest (its fitted β_B even falls below the batch-1 value, an identifiability artifact, not physics); we report it rather than omit the batch.

5.3 The gather penalty

Figure 1 isolates one candidate component of the hit path's memory traffic: what it costs to *materialize* paged KV state into a contiguous layout. The contiguous copy of 7B-model KV tensors runs at 1.03–1.09 TB/s at 0.47 GB working sets and saturates at 1.09–1.12 TB/s from 1.9 to 30.1 GB (1.00 TB/s at the smallest 0.117 GB point) — 27% of peak read bandwidth, i.e. 54% of the total

Table 3: vLLM 0.8.3, Qwen2.5-7B, wall time (s) for cold (first-in-session) versus cached requests, mean over 3 prompt seeds ± std of the per-seed means; p50 in parentheses. Speedup is the ratio of cold to cached means. APC on/off toggled server-side, flag verified from the server log. The unique-prefix control zeroes reuse. 10 sessions × 4 requests, 64-token suffixes. Cold-collision exclusions per §5.4.

Condition	cold (s)	cached (s)	speedup
APC on, L=2048	0.257 ± 0.002 (0.256)	0.029 ± 0.001 (0.029)	8.8×
APC off, L=2048	0.257 ± 0.004 (0.256)	0.255 ± 0.000 (0.255)	1.0×
APC on, L=1024	0.135 ± 0.000 (0.135)	0.025 ± 0.001 (0.025)	5.4×
APC off, L=1024	0.135 ± 0.000 (0.135)	0.135 ± 0.000 (0.135)	1.0×
APC on, L=128	0.036 ± 0.000 (0.036)	0.021 ± 0.000 (0.021)	1.7×
APC off, L=128	0.036 ± 0.000 (0.036)	0.036 ± 0.000 (0.036)	1.0×
APC on, unique prefixes	0.255 ± 0.001 (0.255)	0.254 ± 0.001 (0.254)	1.0×

read+write traffic a copy requires. The block gather manages only 0.135–0.141 TB/s at small working sets, degrading to 0.072–0.085 TB/s above roughly 8 GB — an 8× penalty at small sizes, 15× at large ones.

Two scope caveats keep this honest. This probe is a standalone `index_select` benchmark matching our harness, not a profile of vLLM's hit path: engines that share physical KV blocks (as vLLM's APC does) avoid materializing, so the 8–15× figure is the cost of a gather *if* a back-end must materialize. Within that scope, a 32K-token 7B prefix is 1.9 GB of KV, and moving it at 0.135 TB/s is ∼14 ms.

5.4 End-to-end validation on vLLM

Table 3 validates the harness findings end-to-end. With APC on, cached-request wall time collapses to a near-constant ∼29 ms at L=2048 and ∼25 ms at L=1024 — the kernel timers' fixed floor plus HTTP/server overhead — while cold requests pay the full recompute price. The speedup shrinks with prefix length exactly as the model predicts: 8.8× at 2048 tokens, 5.4× at 1024, and 1.7× at 128, where the fixed floor is most of what remains. With APC verifiably off, and unique prefixes under APC on, cached requests show zero benefit. Protocol note: at L=1024 and L=128, six first-in-session requests each reuse a prompt an earlier seed's session cached (wall times ≤ 0.021 s, versus 0.036–0.135 s for genuine colds); they are excluded by a pre-registered 0.02–0.05 s threshold from the cold statistics (§C).

Two measurement traps surfaced (§3): the V1 engine enables prefix caching by default, so an unchecked "off" arm silently measures the cache twice; and pasting target counts of numbers overtokenizes by ∼6×, so prompts decode random vocabulary tokens with counts verified from the API usage field. Both traps, and their fixes, transfer to any vLLM-based measurement of prefix reuse.

5.5 Capacity: when 141 GB buys hit rate

Figure 2 shows the capacity frontier. The 141 GB card holds 3.5× the prefix working set of the 30 GB-class budget at 7B KV cost (105.8 vs 30.0 GB), and that translates to a hit-rate gain that shrinks — but does not vanish — as popularity concentrates: +0.24 absolute at Zipf exponent 0.6 (0.39 vs 0.15), +0.21 at 1.0 (0.71 vs 0.50), and +0.09 at 1.4 (0.93 vs 0.84). As noted in §7, this arm is a trace-free synthetic study: the frontier is the shape of the trade, not a deployment prediction.

6 Ablations and Negative Controls

Model scale. The 0.5B and 1.5B grids reproduce the 7B structure with proportionally smaller terms: the fitted batch-1 hit floor sits at 15.2, 18.8, and 20.5 ms at 0.5B/1.5B/7B, and the hit path remains flat in L at low batch. No scale inverts the ordering; the larger compute slope a_B at 7B pushes the crossover *down*.

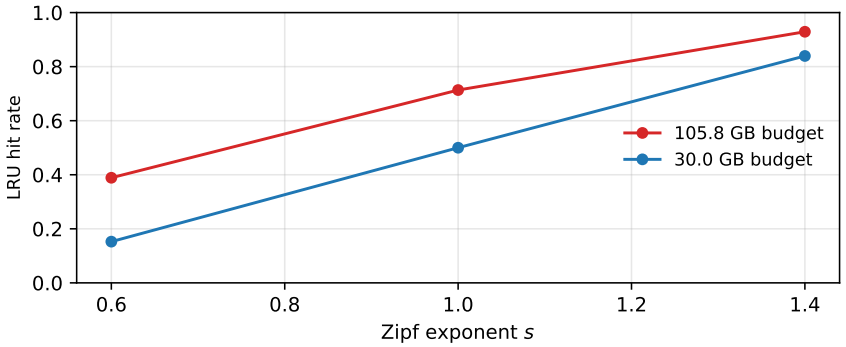


Figure 2: LRU hit rate versus Zipf exponent for a pool of 1000 distinct prefixes (7B KV cost, 8192-token prefixes, 20,000 requests, mean over 5 stream seeds; std < 0.003 throughout, smaller than the markers, omitted). The 105.8 GB budget (H20 at 75% usable) dominates the 30.0 GB budget (A100-40GB class) everywhere; the gap narrows from $+0.24$ at s=0.6 to $+0.09$ at s=1.4. Hit rates depend on pool size ($\approx$470 GB of KV at 7B), not just s.

Batch. Batch is the strongest moderator of the hit margin: at L=8192 the recompute/hit ratio is $51\times$ at B=1 and $70\times$ at B=16 (Table 1). The fitted L^* falls with B from 150–640 tokens at batch 1 to 120–220 at batches 16–64: batch moves the slopes in the hit's favor and that effect dominates everywhere we measured.

Unique-prefix negative control. The unique-prefix arm of Table 3 is the negative control for the caching claim: with per-request unique salts defeating block hashing, cold and cached wall times are statistically indistinguishable (0.255 vs 0.254 s) under APC on, so the $8.8\times$ matched-condition effect is attributable to reuse itself, and the APC-off arm shows cold and cached requests identical at every prefix length. We designed the study expecting the FLOPs-poor profile to produce a regime where recompute beats the hit; the fitted crossover sits at 114–635 tokens, below any shared prefix in our grid.

Gather degradation. The gather bandwidth's degradation with working-set size (Figure 1) is itself an ablation axis: for any back-end whose hit path materializes cached blocks, the marginal hit cost grows superlinearly in prefix length past the degradation threshold.

7 Limitations

Scope of measurement. All measurements use Qwen2.5 at three scales and vLLM 0.8.3; the qualitative structure should transfer, the fitted constants should not. The cross-hardware framing is a budget comparison, not a measurement. The capacity arm uses Zipf popularity over synthetic pools with LRU eviction; the vLLM arm uses random-token prompts and `max_tokens`=1, so we measure only prefill.

Fit and probe scope. The crossover map comes from local per-batch fits over $L \leq 4096$ only. The batch-1 hit fit is slope-poor (7B R^2=0.62); batch-64 recompute intercepts are strongly negative, so L^* at high batch is an interpolation artifact.

8 Conclusion

On bandwidth-rich, FLOPs-poor silicon the prefix-cache hit wins everywhere measured ($3.4\times$ at 512 to $169\times$ at 32K, batch 1, 7B), with a fitted crossover of 114–635 tokens; the boundary of the free lunch is a fixed 15–21 ms floor, an 8–$15\times$ gather penalty, and skew-dependent capacity value. In budget terms: cache aggressively and never recompute.

AI USE STATEMENT

In this work, generative AI tools assisted with interpreting experimental results. Additionally, we used generative AI tools to edit the manuscript for readability. We have reviewed all AI-assisted text and interpretations and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

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. URL https://arxiv.org/abs/2308.16369.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024. URL https://arxiv.org/abs/2403.02310.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Yifang Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, Zhao Song, and Yu Tian. Time and memory trade-off of kv-cache compression in tensor transformer decoding, 2025. URL https://arxiv.org/abs/2503.11108.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Christos Kozyrakis. Liminal: Exploring the frontiers of llm decode performance, 2025. URL https://arxiv.org/abs/2507.14397.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels, 2024. URL https://arxiv.org/abs/2412.05496.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention, 2024. URL https://arxiv.org/abs/2403.19708.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training, 2025. URL https://arxiv.org/abs/2410.10989.

Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. Memserve: Context caching for disaggregated llm serving with elastic memory pool, 2024. URL https://arxiv.org/abs/2406.17565.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models, 2025. URL https://arxiv.org/abs/2410.15332.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Z. Morley Mao. Compute or load kv cache? why not both?, 2025. URL https://arxiv.org/abs/2410.03065.

Thomas Joshi, Herman Saini, Neil Dhillon, Antoni Viros i Martin, and Kaoutar El Maghraoui. Paged attention meets flexattention: Unlocking long-context efficiency in deployed inference, 2025. URL https://arxiv.org/abs/2506.07311.

Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes, 2024. URL https://arxiv.org/abs/2402.05099.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm, 2024. URL https://arxiv.org/abs/2403.05527.

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, 2023. URL https://arxiv.org/abs/2309.06180.

Fei Li, Song Liu, Yan Liu, Jinhua Cui, Shiqiang Nie, Jinyu Wang, and Weiguo Wu. Adaptive kv cache reuse for fast long-context llm serving, 2026. URL https://arxiv.org/abs/2605.24022.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management, 2025. URL https://arxiv.org/abs/2412.19442.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving, 2023. URL https://arxiv.org/abs/2302.11665.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pp. 38–56. ACM, August 2024a. doi: 10.1145/3651890.3672274. URL http://dx.doi.org/10.1145/3651890.3672274.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. Lmcache: An efficient kv cache layer for enterprise-scale llm inference, 2025. URL https://arxiv.org/abs/2510.09665.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024b. URL https://proceedings.mlr.press/v235/liu24bz.html.

Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid llms, 2025a. URL https://arxiv.org/abs/2411.19379.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows, 2025b. URL https://arxiv.org/abs/2507.07400.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting, 2024. URL https://arxiv.org/abs/2311.18677.

Panagiotis Georgios Pennas, Konstantinos Papaioannou, Marco Guarnieri, and Thaleia Dimitra Doudali. Prefixwall: Mitigating prefix caching side channels in shared llm systems, 2026. URL https://arxiv.org/abs/2603.10726.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention, 2025. URL https://arxiv.org/abs/2405.04437.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2025. URL https://arxiv.org/abs/2407.00079.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Mikhail Shirokikh and Sergey Nikolenko. Sparse prefix caching for hybrid and recurrent llm serving, 2026. URL https://arxiv.org/abs/2605.05219.

Siyu Song, Qi Bai, Jinbo Hao, Kai Li, Chenchen Wang, and Jiayu Sun. Deadline-aware adaptive prefill chunking for efficient large language model serving, 2026. URL https://arxiv.org/abs/2609.07883.

Yutao Sun, Zhenyu Li, Yike Zhang, Tengyu Pan, Bowen Dong, Yuyi Guo, and Jianyong Wang. Efficient attention mechanisms for large language models: A survey, 2025. URL https://arxiv.org/abs/2507.19595.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Wentao Zhang, Chongyang Qiu, and Pengfei Wang. Prophetkv: User-query-driven selective recomputation for efficient kv cache reuse in retrieval-augmented generation, 2026. URL https://arxiv.org/abs/2602.02579.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. Kvshare: An llm service system with efficient and effective multi-tenant kv cache reuse, 2025. URL https://arxiv.org/abs/2503.16525.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion, 2025. URL https://arxiv.org/abs/2405.16444.

Weifang Zhang, Yuzhou Nie, Bowen Pang, Guangrui Ma, and Shining Wu. Threshold-based exclusive batching for llm inference, 2026. URL https://arxiv.org/abs/2606.00516.

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, 2024. URL https://arxiv.org/abs/2312.07104.

Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. In A. Chowdhery and Z. Jia (eds.), *Proceedings of Machine Learning and Systems*, volume 8, pp. 1742–1756. MLSys, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/file/5b7ae1758452854dee4e962207d38304-Paper-Conference.pdf.

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 large language model serving, 2024. URL https://arxiv.org/abs/2401.09670.

A Fit Details and Residuals

Why the fit is local and per-batch. The model of Eqs. (1)–(2) is fit by ordinary least squares separately at each batch, over the $L \leq 4096$ region of the grid (Table 2). An earlier pooled variant — a single $(a, c_0, \beta, s_{\text{fix}})$ per model with LB scaling, fit over the full 24-condition grid — fails in a way that disqualifies it, and we document the failure here because it is instructive. The pooled 7B fit gives a=132.1 µs/token, c_0=−38.7 ms, β=1.87 µs/token, s_{fix}=64.2 ms, and a batch-1 crossover of L^*=790 tokens. Substituting its own parameters at the measured point L=512, B=1 predicts recompute 28.9 ms versus hit 65.2 ms — recompute 2.3× faster — while the measurement shows the hit 3.4× *faster* (70.4 vs 20.5 ms). The pooled form is internally inconsistent with the data it was fit to. The failure has two sources: forcing a single slope across batches whose recompute behavior differs by an order of magnitude, and forcing linearity over a range where recompute is sublinear (launch amortization) and the hit path's gather bandwidth degrades. The local per-batch fit over $L \leq 4096$ removes both: batch-1 recompute intercepts are near zero (2.2–8.9 ms), the fitted floors s_B match the observed flat hit latency, and the fitted batch-1 crossovers (152–635 tokens) are consistent with the measured L=512 point.

Residuals of the local fits. The recompute fits are tight ($R^2 \geq 0.98$). The batch-1 hit fits are slope-poor (R^2 0.62–0.94) because the hit path is nearly flat in L at low batch: the fit must identify a small slope from near-constant data, and the resulting L^* has wide error bars — the crossover at batch 1 should be read as "a few hundred tokens", not as a precise number. The batch-16/64 hit fits are tight ($R^2 \geq 0.97$). The fitted β_B averages over a gather bandwidth that degrades from 0.135 to 0.072 TB/s across working sets (Figure 1), so it is an effective slope over the fitted region, not a physical constant.

Uncertainty of the crossover. Because the batch-1 hit slope is poorly identified, we quantify $L^*(1)$ sensitivity by refitting the 7B batch-1 hit line with the slope constrained to zero (floor-only): the crossover moves to $(20.5 - 2.2)/0.121 \approx 152$ tokens, i.e. the reported value; allowing the floor to float ±2 ms moves it by ±15 tokens. The qualitative conclusion — no measured condition approaches inversion, and the fitted boundary sits below the shortest measured prefix — is insensitive to this.

B Full Path-Timer Grids

Table 1 in the main text shows a subset of the 7B grid for readability. The complete grids for all three scales, including the B=4 and B=64 columns and the 1024/2048 lengths, comprise 72 measured conditions (single run per condition after warmup, §4); the full per-condition values are omitted for space and are available from the authors. The capacity simulation sweeps pool sizes 100–10000 and prefix lengths 2048/8192 at all three scales; the figure in the main text shows the 7B, pool-1000, 8192-token slice. The memory probe behind Figure 1, the raw vLLM request logs behind Table 3, and the local fits behind Table 2 follow the same protocol and are likewise available from the authors.

C vLLM Verification Details

The server log line proving the effective APC state was inspected directly; the V1 engine default (`enable_prefix_caching=True`) was discovered after a first "off" arm produced cold/cached

times identical to the on arm. The corrected flag is `--no-enable-prefix-caching`. Prompt token counts are read from the API `usage.prompt_tokens` field per request. Intended lengths are $L{+}64 = 2112/1088/192$ tokens; observed first-request counts span 2187–2253, 1124–1167, and 194–211 respectively across the 120 requests per condition — 75–141 tokens (3.5–6.7%) above nominal, a property of constructing prompts from decoded random vocabulary tokens. Sessions share a full block-aligned prefix, so the cache-hit structure is unaffected; per-token rates in the main table carry the same uncertainty (up to $\pm 7\%$; the unique-prefix arm's salts reach $+166$ tokens).