

OpenDC-Infer-LC: Benchmarking Production-Valid Long-Context and RAG LLM Serving

Yue Yang^{*1}, Maxwell Twelftree^{*1}, Jared Collette¹, An-chi He¹,
Luke Borgnolo¹, David Lemphers¹,

¹Maincode, Melbourne, Australia,

Correspondence: yue@maincode.com

Abstract

Long-context and RAG workloads expose production bottlenecks that raw tokens/s and offline accuracy do not capture, including pre-fill cost, KV-cache pressure, prefix reuse, routing, and latency SLOs. We introduce OpenDC-Infer-LC, a hardware- and backend-neutral benchmark for evaluating complete data-center LLM serving stacks under long-context and retrieved-context workloads. The benchmark ranks systems by SLO-qualified, quality-preserving goodput: valid output-token throughput at the highest load satisfying p95 TTFT, p95 TPOT, p95 end-to-end latency, and success-rate targets. Because RAG and Long-Bench quality depends on model ability, retrieval difficulty, and task difficulty, OpenDC-Infer-LC uses a relative quality guardrail against a low-load reference run for the same model and workload, while reporting absolute quality separately. Pilot results on AMD MI355X and NVIDIA B200 show that deployable capacity depends strongly on backend scheduling, cache reuse, context length, routing, quantization, and runtime maturity.

1 Introduction

Long-context and retrieval-augmented generation (RAG) workloads are now central to production language-model systems. Enterprise assistants, codebase copilots, legal and scientific document tools, customer-support agents, and multi-turn analytic systems increasingly send tens or hundreds of thousands of input tokens to a model while still requiring streamed responses within interactive latency budgets (Li et al., 2024b; Liu et al., 2024). These workloads expose bottlenecks that are largely absent from short chat completion: pre-fill can dominate end-to-end latency, KV-cache capacity can limit concurrency, prefix reuse can change effective throughput by orders of magnitude, and multi-node serving performance can depend on scheduler behavior, container configura-

tion, storage paths, routing policy, and accelerator-fabric communication (Chen et al., 2024; Chkribene et al., 2024; Li et al., 2024a; Twelftree et al., 2026). However, existing evaluation methods do not provide a complete answer to this production problem. Model-centric long-context benchmarks test whether a model can understand or retrieve information from long inputs, but they do not measure whether a deployed serving stack can sustain such workloads at scale (Bai et al., 2023; Hsieh et al., 2024; Liang et al., 2023). General inference benchmarks and serving profiles (MLCommons, 2025; NVIDIA, 2025) measure throughput and latency, but they usually focus on endpoint-level performance under generic request patterns and do not fully specify cache state, prefix reuse, load-generator validity, multi-node transport behavior, or quality-preserving SLO compliance (Vinay, 2025). As a result, they can report systems that appear fast in raw tokens per second but are unusable in practice because they violate p95 latency targets, depend on hidden warmed caches, saturate the client-side load generator, fall back to slower transport, or produce degraded answers.

This leaves a practical gap for industry deployment: **teams need to know not only how fast a backend can generate tokens, but how much correct long-context traffic a complete data-center serving system can reliably sustain under a heavy workload, model, tokenizer, cache state, and service-level objective.** To address this, we define **OpenDC-Infer-LC**, a hardware- and backend-neutral benchmark for long-context and RAG inference serving in data-center environments. Unlike MLPerf Inference (MLCommons, 2025), which provides broad, architecture-neutral inference benchmarks for standardized models and scenarios, OpenDC-Infer-LC focuses specifically on production long-context and RAG serving. It treats the full serving path as the object of evaluation, including scheduler allocation, serving runtime, ac-

Table 1: Positioning of OpenDC-Infer-LC against existing benchmark families.

Benchmark family	Main focus	LC	Cache	System	SLO	Quality
MLPerf / GenAI-Perf	Generic inference throughput and latency	P	N	P	P	P
LongBench / RULER / NIAH	Long-context model ability	Y	N	N	N	Y
HELM / RAGAS / BEIR / KILT / ARES	Model, retrieval, or grounded-answer quality	P	N	N	N	Y
Backend-specific studies	Runtime or kernel performance	P	P	P	P	P
OpenDC-Infer-LC	Full LC/RAG serving capacity	Y	Y	Y	Y	Y

Y = yes; P = workload-dependent(not for Long-context); N = not a primary benchmark feature.

celerator communication, cache placement, storage paths, telemetry, endpoint behavior, and load generation. The benchmark reports *SLO-qualified goodput*: output-token throughput from successful requests at the highest offered load that satisfies run-level p95 time-to-first-token, p95 time-per-output-token, p95 end-to-end latency, and success-rate constraints. OpenDC-Infer-LC defines a suite of long-context serving workloads. The core benchmark includes LC-8K for standard RAG, LC-32K for enterprise document QA, LC-128K for heavy long-document and codebase QA, and LC-Cache for repeated-prefix and multi-turn RAG. To better capture realistic RAG behavior beyond synthetic long-context prompts, we further include RAG-oriented workloads: RAG-8k, 32K, 128K, which evaluates serving under top- k retrieved-passage inputs. For each query, deterministic BM25 ranks the shared 66,635-passage corpus. We greedily pack top-ranked passages to 8K, 32K, and 128K token budgets and record whether the gold supporting passages survive within the packed context. We also tested the classic long-bench with our proposed metric.

- we define OpenDC-Infer-LC, a production-oriented benchmark for long-context and RAG inference serving that evaluates complete serving systems rather than isolated model endpoints;
- we introduce SLO-qualified, quality-gated goodput, together with cache-state, endpoint-measurement, load-generation, reliability, transport/storage, and hardware-efficiency rules for reproducible capacity evaluation;
- we release a reference harness and present pilot studies across diverse hardware and serving configurations, showing that raw throughput can differ from production-valid serving capacity.

2 Related Work

OpenDC-Infer-LC is closest to system-level inference benchmarks and long-context model evaluations, but targets a different measurement bound-

ary. MLPerf Inference and GenAI-Perf standardize LLM serving metrics such as throughput, time to first token, and per-token latency, and are important references for deployment-oriented evaluation (MLCommons, 2025; NVIDIA, 2025). Our benchmark complements them by focusing specifically on long-context and RAG-style data-center serving, with explicit cache-state rules, prefix-reuse workloads, storage and interconnect reporting, multi-node qualification, and SLO-qualified goodput. Long-context benchmarks such as LongBench, RULER, and Needle-in-Haystack evaluate whether a model can use long inputs correctly (Bai et al., 2023; Hsieh et al., 2024; Liang et al., 2023). Similarly, holistic and RAG-oriented evaluation frameworks such as HELM, RAGAS, BEIR, KILT, and ARES focus on model behavior, retrieval quality, grounded-ness, or answer quality (Liang et al., 2023; Es et al., 2024; Thakur et al., 2021; Petroni et al., 2021; Saad-Falcon et al., 2023). These benchmarks are useful for selecting or validating models, but they generally do not measure whether a complete serving stack can sustain long-context traffic under production latency and reliability constraints.

In real industry deployments, the key question is not how fast a backend can generate tokens in isolation, but how much correct long-context traffic the full serving system can sustain under production constraints (Bertsch et al., 2025; Bai et al., 2025). A system with high raw throughput is not useful if its p95 time-to-first-token violates the latency contract, since users will experience the service as slow or broken. Similarly, optimizations such as quantization, aggressive batching, prefix caching, or routing are only valuable if they preserve answer correctness, especially in enterprise QA, legal, finance, codebase assistants, and customer-support scenarios (Chkibene et al., 2024; Li et al., 2024a). Industry teams also need reliable capacity and cost planning: they must know how many users a cluster can support before violating SLOs, which raw throughput alone cannot reveal. Moreover, fair hardware comparison requires exposing workload-specific

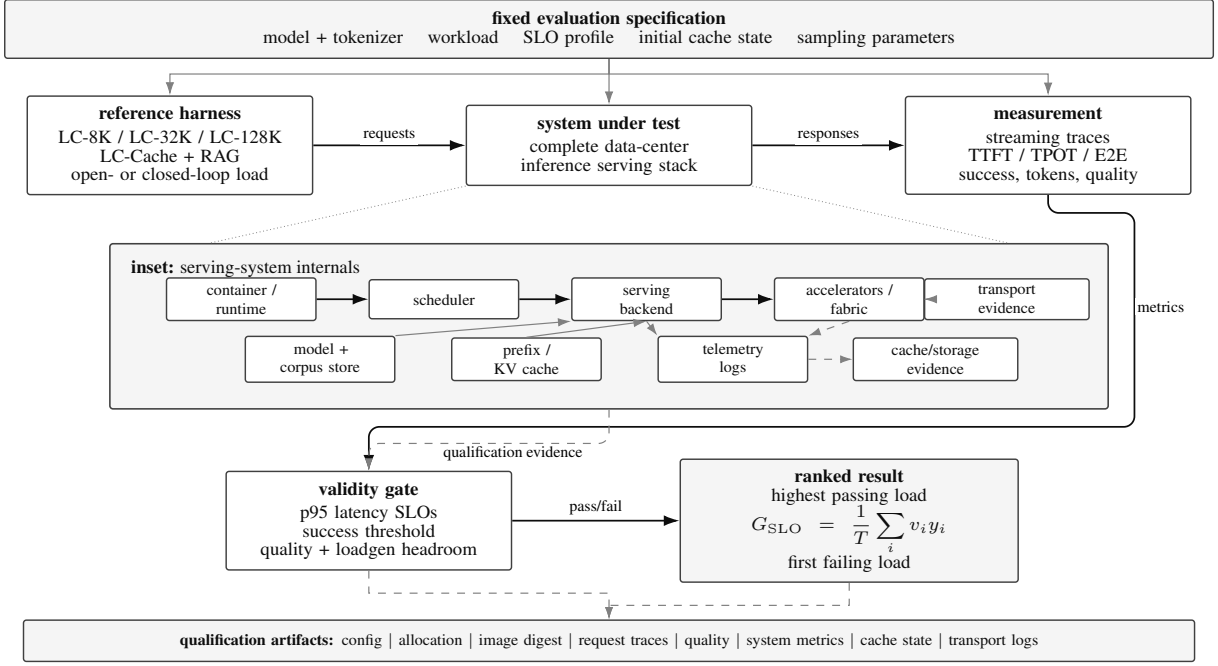


Figure 1: OpenDC-Infer-LC benchmark pipeline. A fixed evaluation specification defines the model and tokenizer, workload, SLO profile, initial cache state, and sampling parameters.

bottlenecks, such as whether a GPU is better for short-context decode or cached RAG due to HBM and KV-cache capacity. By enforcing cache-state rules, endpoint measurement, load-generator validity, token accounting, and quality guardrails, the benchmark prevents misleading results caused by warmed caches, incomplete accounting, client bottlenecks, or failed requests being counted. Therefore, the core value of this mechanism (Table 1) is that it shifts evaluation from “How fast can this backend generate tokens?” to “How much correct long-context traffic can this complete serving system sustain under real production constraints?”.

3 Methodology

We define **OpenDC-Infer-LC**, an open, hardware-neutral benchmark for long-context and retrieval-augmented generation (RAG) inference serving in data-center environments. It complements model-centric long-context evaluations and generic throughput benchmarks: rather than reporting peak tokens per second, OpenDC-Infer-LC measures whether a serving system can sustain long-context workloads under realistic service-level objectives (SLOs), and additionally reports cache behavior, multi-node capacity and reliability. The central question for us is:

“Given a fixed model, tokenizer, workload, and latency requirement, how many

long-context or RAG requests can a cluster serving system process reliably?”

This differs from endpoint-only benchmarking. OpenDC-Infer-LC evaluates the full serving system—scheduler, container runtime, accelerator allocation, interconnect, storage and cache paths, telemetry, serving backend, and request load generator—to measure production-relevant capacity rather than whether an endpoint is merely reachable. We target long-context and RAG-style inference because these workloads dominate a growing share of production traffic (enterprise document QA, codebase reasoning, agent memory, multi-turn retrieval-augmented assistants) and, relative to short chat, place far greater pressure on prefill efficiency, KV-cache memory, request scheduling, prefix reuse, and multi-node scaling. The benchmark is hardware- and backend-neutral: any accelerator and any serving backend may participate, provided it follows the measurement and submission rules.

3.1 Benchmark Design

OpenDC-Infer-LC defines eight v1.0 workloads across synthetic long-context, repeated-prefix cache, RAG, and LongBench-style real-task evaluation. Each fixes a target input length, a maximum output length, and a target serving scenario. It exercises three workload families through one common harness: synthetic needle-in-a-haystack long-

context, the real LongBench task suite. To evaluate retrieval-centric long-context serving, OpenDC-Infer-LC includes RAG-TopK, a corpus-backed retrieved-context workload built from public available QA dataset (Yang et al., 2018). We pool and deduplicate all distractor-split passages into a shared 66,635-passage corpus, retrieve answer-bearing and distractor passages for each query, and pack the ranked passages to 8K, 32K, and 128K token budgets. The resulting prompts are frozen per tokenizer so that every backend and hardware stack is evaluated on identical retrieved contexts. This design isolates the serving-system question: given realistic retrieved-passage inputs, how much RAG-shaped traffic can the LLM serving stack sustain under SLOs? Retrieval recall, gold-in-context status, and answer F1/EM are reported as task diagnostics, while SLO-goodput measures timed generation serving over the standardized retrieved contexts. The public corpus for LC-X is **synthetic and seed-generated**, using a RULER-style needle-in-a-haystack template with “magic number” answers for exact-match checking. This design avoids license issues, prevents real-document reuse from artificially inflating prefix-cache metrics, and enables bit-for-bit reproducibility. Prompts are tokenizer-specific, length-controlled to the target budget (≤ 5 -token drift), and pinned by a `dataset_version_`-hash; Generation details are in Appendix C.1. To evaluate retrieval-centric long-context serving, we construct frozen RAG splits from a QA distractor set. Rather than retrieving from each question’s small candidate set, we pool and de-duplicate all distractor passages into a shared 66,635-passage corpus, then use a deterministic BM25 retriever to rank the full corpus for each query. Top-ranked passages are packed to model-tokenizer budgets of 8K, 32K, and 128K, producing RAG-8K, RAG-32K, and RAG-128K workloads over increasingly large retrieved contexts. Each split records retrieval metrics such as recall@K and MRR, along with gold-in-context indicators, enabling us to separate retriever failures from generator failures while running RAG through the same serving harness as the synthetic long-context workloads. See Appendix C.1 for generation details.

3.2 System-level Evaluation Metrics

OpenDC-Infer-LC scores each run using one primary leaderboard metric, *quality-gated SLO-goodput*, together with secondary diagnostics (Appendix C.2).

SLO qualification. A run is *SLO-qualified* if its aggregate p95 TTFT, p95 TPOT, p95 E2E latency, and success rate satisfy the selected SLO profile over the full measurement window (see Table 6 in appendix for details). Runs that violate the latency or reliability SLO are ineligible for ranking regardless of raw throughput.

Quality guardrail. The quality guardrail prevents systems from increasing throughput by degrading model outputs. We distinguish absolute task quality from relative quality preservation. For each workload w , the harness computes an absolute task score $Q_{\text{abs}}(s, w)$ for system s . The scoring function is workload-specific: synthetic LC tasks use exact-match accuracy against the generated needle value; LC-Cache uses exact match together with cross-turn consistency; RAG workloads report answer against the gold answer over the frozen retrieved contexts; and LongBench uses the official task-specific metrics and macro average. Because RAG and LongBench absolute scores depend on model capability, retrieval difficulty, and the frozen prompt set, leaderboard validity is based on quality preservation rather than on an absolute score of 0.98. For each model m , tokenizer, workload, prompt set, and sampling policy, we define a reference quality score $Q_{\text{ref}}(m, w)$ from a low-load reference run. A submitted system is quality-valid if

$$Q_{\text{rel}}(s, w) = \frac{Q_{\text{abs}}(s, w)}{Q_{\text{ref}}(m, w)} \geq \tau_w^Q.$$

In OpenDC-Infer-LC v1.0, the default quality-preservation threshold is $\tau_w^Q = 0.98$. Appendix H Table 14 reports Q_{ref} , Q_{abs} and Q_{rel} , and pass/fail status for the reported quality-valid runs, clarifying that low absolute RAG scores are ranked only when they preserve the low-load reference quality for the same model, workload, tokenizer, and sampling policy. The relative guardrail is intentionally a serving-stack validity test, not a universal task-quality threshold. For RAG and LongBench, absolute quality depends on model capability, retriever difficulty, context packing, and task difficulty. A low absolute score may therefore indicate that the model or retriever is weak on the workload, not that the serving stack degraded the output. OpenDC-Infer-LC separates these cases by ranking only runs that preserve the low-load reference quality. Users who require application-level acceptance can impose an additional workload-specific Q_{abs} threshold on top of the serving-valid result.

3.2.1 Quality-gated SLO-goodput.

Quality preservation is evaluated at the run/workload level. For a candidate run r on workload w , let $q_r = \mathbf{1}\{Q_{\text{rel}}(r, w) \geq \tau_w^Q\}$ be the run-level quality-validity indicator, and let $s_r = \mathbf{1}\{\text{TTFT}_{95}, \text{TPOT}_{95}, \text{E2E}_{95}, \text{success}\}$ satisfy the selected SLO; be the run-level SLO-validity indicator. Let $d_i = 1$ if request i successfully completes, is not timed out or cancelled, and produces a well-formed final output; otherwise $d_i = 0$. For a measurement window of duration T , the ranked goodput of a valid run is

$$G_{\text{SLO}}(r) = \frac{1}{T} \sum_{i \in R} d_i y_i.$$

A run is leaderboard-valid only when $s_r q_r = 1$. If either the SLO gate or the run-level quality-preservation gate fails, the run is not ranked; its raw throughput may be reported only as a diagnostic.

Capacity search and scaling: During capacity search, the harness increases the offered load until the SLO is violated. We report the highest leaderboard-valid load point and retain the first violating point as the saturation boundary. For multi-node submissions, we report scaling efficiency as $\text{Efficiency}(N) = \frac{G(N)}{N \cdot G(1)}$, where N is the node count and G is $\text{Goodput}_{\text{SLO}}$. If a configuration fails the quality guardrail uniformly across the sweep, we additionally report raw SLO-qualified output throughput as a marked diagnostic, but such results are excluded from leaderboard ranking.

Load Generation: OpenDC-Infer-LC supports both closed- and open-loop load generation. In closed-loop mode, the harness sweeps concurrency (1, 2, 4, 8, ...) until an SLO fails. In open-loop mode, it sweeps a Poisson arrival rate to saturation. For each workload, we report the highest SLO-qualified load and retain the first failing point as the un-ranked saturation boundary (Appendix C).

4 Experiment Results and Analysis

4.1 Experimental Setup

We instantiate OpenDC-Infer-LC (Section 3) on two data-center accelerators and the two MoE tiers defined there—Qwen3-235B-A22B (headline) and Qwen3.5-35B-A3B (medium). All runs use the reference harness and the frozen, hash-pinned prompt sets; the harness, datasets, and one-file cluster runners are released for reproduction. All ranked pilot points passed the harness validation checks: load-generator headroom, complete request labeling,

token-accounting checks, streaming-timing checks, and clock-source checks.

Platforms and backends. We use the same vendor-neutral harness on two platforms, varying only the serving and launch layer. **AMD MI355X** (CDNA4/gfx950, 288 GB HBM3E/GPU) is our primary platform, with 8 GPUs/node and up to 8 SLURM nodes served via rootless podman and ROCm. **NVIDIA B200** (Blackwell, ~192 GB HBM3E/GPU) is evaluated on one 8-GPU node via Docker and CUDA for single-node cross-vendor comparison.

Multi-node. For scale-out, we run one TP=8 replica per node behind a single sg1-router endpoint and sweep $N \in \{1, 2, 4, 8\}$ nodes (up to 64 MI355X GPUs). We run one full model replica per node, place an OpenAI-compatible router in front of all N replicas, and sweep four routing strategies — round-robin(RB), least-outstanding-requests(LO), prefix-hash (cache-affinity, PH), and hybrid (cache-affinity with load-balanced fallback, Hyb) — over a multi-prefix cache workload. For each we report SLO-qualified goodput, latency tails, and scaling efficiency.

4.2 Results and Analysis

Table 2 reports single-node SLO-qualified goodput for both models on both vendors; Table 3 reports multi-node scaling; Appendix I gives representative load sweeps, and the full per-load-point CSVs, request traces, quality outputs, and system logs are included in the artifact. Our pilot is not designed to isolate accelerator-only effects. The stacks differ in runtime, container, driver, and backend maturity. We therefore interpret these results as comparisons of complete serving configurations rather than as a hardware leaderboard. Under this framing, the results show that the highest SLO-qualified goodput changes with workload, model size, backend, cache behavior, and routing policy. These results motivate a benchmark that reports the full evaluated serving configuration rather than a single peak-throughput number.

Insight 1: Why not raw throughput? Raw tokens/s overstates deployable capacity because it counts work that a production service cannot actually use. Appendix E shows four measured failure modes. First, raising offered concurrency can make the endpoint look faster while breaking the latency contract: on lc_8k with SGLang-35B, increasing C from 8 to 16 raises raw output throughput by 61% (83.2→133.9 tok/s), but TPOT₉₅ crosses the

Table 2: **Main single-node results** on AMD MI355X and NVIDIA B200 (8 GPUs/node each), grouped by model and platform; *fail*_{SLO} denotes that no offered load satisfied the latency/reliability SLO. For each entry, we have N_c , where N is the SLO goodput, and c the highest concurrency that still meets every SLO;

Model Workload		NVIDIA B200		AMD MI355X	
		vLLM BF16	SGLang BF16	vLLM BF16	SGLang BF16
235B A22B TP8	LC-8K	36.4 ₄	38.3 ₄	22.2 ₂	31.3 ₄
	LC-32K	7.2 ₁	9.1 ₂	3.6 ₁	6.5 ₁
	LC-128K	1.2 ₁	1.3 ₁	<i>fail</i> _{SLO}	0.9 ₁
	LC-Cache	64.6 ₁₆	63.8 ₈	47.5 ₈	99.6 ₁₆
	RAG-8K	12.8 ₂	6.0 ₁	4.0 ₁	1.27 ₁
	RAG-32K	3.1 ₁	1.77 ₁	1.77 ₁	2.55 ₁
	RAG-128K	0.3 ₁	<i>fail</i> _{SLO}	<i>fail</i> _{SLO}	0.22 ₁
	LongBench	49.4 ₈	1.33 ₁	1.27 ₁	1.58 ₂
35B A3B TP8	LC-8K	106.6 ₃₂	61.6 ₈	39.67 ₄	78.98 ₈
	LC-32K	27.1 ₈	18.2 ₂	7.41 ₁	19.25 ₂
	LC-128K	6.8 ₈	2.0 ₁	<i>fail</i> _{SLO}	3.21 ₁
	LC-Cache	26.8 ₈	52.4 ₈	23.8 ₄	73.4 ₁₆
	RAG-8K	31.1 ₄	22.0 ₂	13.97 ₂	7.8 ₁
	RAG-32K	10.9 ₈	4.7 ₁	3.17 ₁	6.22 ₁
	RAG-128K	2.9 ₈	0.9 ₁	<i>fail</i> _{SLO}	0.93 ₁
	LongBench	70.3 ₃₂	56.2 ₈	2.3 ₁	25.3 ₂

interactive cap ($0.124s > 0.12s$), so the ranked capacity remains the last valid point, $C=8$, at 79.0 tok/s. Second, cache-heavy optimizations can be fast but wrong: on `lc_cache_mp`, prefix reuse yields 102.1 out-tok/s and 738K total tok/s, yet quality is 0.00, so quality-gated goodput is zero. Third, very long contexts may emit tokens without being serviceable: vLLM-35B produces throughput on RAG_128k and lc_128k, but misses the decode-latency caps, leaving no SLO-qualified point. Finally, multi-node routing shows that cache locality is not sufficient: at $N=8$, prefix-hash remains near 7.4 tok/s ($E=0.12$), while round-robin reaches 50.4 tok/s ($E=0.85$). These cases motivate G_{SLO} : the metric counts only output that is fast, reliable, and correct at the highest valid load.

Insight 2: Long-context and RAG workloads expose the main SLO bottlenecks. Goodput drops sharply as context length increases, and the highest valid concurrency usually falls at the same time. For the 235B model, the best LC goodput falls from 38.3 on LC-8K to 1.3 on LC-128K, while the best RAG goodput falls from 12.8 on RAG-8K to 0.3 on RAG-128K. The 35B model is more robust, but the decline remains substantial: LC drops from 106.6 to 6.8, and RAG drops from 31.1 to 2.9 when moving from 8K to 128K. This shows that short-context throughput is not a reliable proxy for long-context or RAG serving performance and deployment need be evaluated directly at the intended context length and concurrency.

Insight 3: Load-balanced routing is the right default for multi-node serving. Table 3 shows

a clear tension between cache affinity and cluster-level scaling. Round-robin and least-outstanding-requests both scale well as the number of MI355X nodes increases, growing from about 7.4 tok/s on one node to 50.4–50.6 tok/s on eight nodes. Their eight-node efficiencies remain high at 0.85–0.86, showing that the system still benefits strongly from adding replicas. The small loss from ideal linear scaling is expected in multi-node serving, where routing overhead, queue imbalance, and tail-latency effects become more visible. In contrast, prefix-hash and hybrid routing preserve cache locality but fail to scale goodput. Both remain near the single-node level of ≈ 7.4 tok/s even as the cluster grows to eight nodes, so efficiency drops roughly as $1/N$. This indicates that hot prefixes are being pinned to specific replicas, leaving other nodes underused. For this multi-prefix cache workload, maximizing KV-cache reuse is therefore less important than distributing work evenly across replicas.

4.3 Mixed-Traffic and Production-Trace Evaluation

The standard G_{SLO} definition gates run-level p95 measurements against a single workload profile. A mixed-traffic run therefore has no unique run-level gate: each request class has a different latency profile, and every realistic mixed arm would otherwise be reported as “not ranked.” For this analysis, we retain the default workload-specific latency bounds and quality gate from the main paper, but apply them per request rather than per run. Each request receives credit 1 if it meets its class-specific deadline, $1/x$ if it is late by a factor of x ,

Table 3: Multi-node routing $\times$ scaling on the multi-prefix cache workload; Each cell is SLO-qualified goodput in tok/s with scaling efficiency $E(N)$ in parentheses; $E = 1.00$ is ideal linear scaling.

Method	$N=1$	$N=2$	$N=4$	$N=8$
RB	7.4 (1.00)	14.3 (0.97)	27.5 (0.93)	50.4 (0.85)
LO	7.3 (1.00)	14.5 (0.99)	27.5 (0.94)	50.6 (0.86)
PH	7.4 (1.00)	7.3 (0.50)	7.4 (0.25)	7.4 (0.12)
Hyb	7.3 (1.00)	7.3 (0.50)	7.4 (0.25)	7.4 (0.13)

Table 4: Performance under realistic mixed-traffic arrival processes. Confidence intervals are 95% intervals.

Traffic arm	Raw tokens/s	$G_{\text{SLO}}^{\text{eq}}$ (95% CI)	Change vs. mixed Poisson	Qualified/raw
Mixed, Poisson	1.696	1.394 [1.343, 1.439]	—	82.2%
Mixed, bursty	1.808	1.007 [0.944, 1.071]	−27.8%	55.7%
Production-trace replay	1.400	0.514 [0.485, 0.543]	−63.1%	36.7%

and 0 if it is dropped. Credits are averaged within each request class and combined across classes using a compute-weighted geometric mean. The denominator is fixed to the offered schedule and gold-answer lengths, so request shedding or output padding cannot increase the score. On a qualifying single-workload run, the resulting equivalent score, $G_{\text{SLO}}^{\text{eq}}$, is exactly equal to G_{SLO} ; in our validation case, both are 1.3028.

Setup. We replay the Apache-2.0-licensed Mooncake FAST’25 production conversation trace, which contains 12,031 requests over 3,537 seconds, on the paper’s serving stack: vLLM in BF16, Qwen3-235B-A22B, tensor parallelism of 8, and eight AMD MI355X GPUs. The trace is replayed at 0.25 requests/s using 1,800-second measurement windows. The arrival process and length mixture are measured from the trace rather than assumed. The inter-arrival coefficient of variation is 3.03, compared with 1.0 for the homogeneous Poisson mode reported in the final row of Table 9 of the main paper, and the request mix is 77.3% LC-8K, 20.6% LC-32K, and 2.1% LC-128K. Every point uses a freshly restarted server. Load-generator issue lateness remains at or below 35 ms at p95, satisfying the headroom rule in Section C.3 of the main paper.

Results. When served independently at their in-mixture arrival rates, LC-8K and LC-32K qualify, with $G_{\text{SLO}} = 1.30$ and 0.32 tokens/s, respectively. LC-128K fails even when served alone, consistent with the FAILSLO result in Table 2 of the main paper. Interleaving exactly these requests at exactly these rates produces no qualifying load point; under the definition in Section 3.2.1, G_{SLO} is therefore undefined for every realistic mixed-traffic arm.

The main mechanism is head-of-line blocking. LC-8K TTFT p95 increases from 0.34 s when served alone to 10.45 s under mixed Poisson traffic and 30.52 s under the production-trace timing, an $89\times$ increase relative to the single-class case. This occurs even though the mixed workload offers less than half the token load of the homogeneous control, approximately 3,960 versus 8,190 tokens/s. The degradation is therefore caused by service-time heterogeneity rather than aggregate offered load.

Public anonymised traces provide timestamps and token counts rather than prompt text, and are dominated by shorter contexts. The Azure 2024 trace contains a maximum context length of 7,999 tokens over 44 million requests, whereas the Mooncake trace reaches 126,195 tokens. We therefore replay only the arrival process and observed length mixture over the frozen benchmark corpus, and make no claim beyond 128K context. LC-128K represents 2.1% of the trace, corresponding to 5–14 requests per measurement window.

5 Conclusion

We introduced OpenDC-Infer-LC, a hardware- and backend-neutral benchmark for long-context and retrieved-context RAG serving. Instead of ranking peak tokens/s, we measure SLO-qualified, quality-gated goodput: the rate of correct output a complete serving stack can sustain under latency, reliability, and workload constraints. Our pilot across AMD MI355X and NVIDIA B200 systems shows that deployable capacity depends strongly on workload, backend, cache behavior, context length, and routing policy. These results support the central message that long-context serving systems should be compared by production-valid capacity.

Limitations

OpenDC-Infer-LC is a serving-capacity benchmark, not a universal hardware ranking. Our pilot reflects the current maturity of the evaluated backend, driver, quantization, and routing stacks; some configurations remain coverage gaps. Due to the resources constraints, B200 results are single-node, while MI355X includes the multi-node study, so cross-vendor comparisons should be read as system-stack comparisons. RAG workloads use frozen retrieved-context prompts to standardize evaluation; OpenDC-Infer-LC does not certify that a model or retriever is application-acceptable; it certifies that the serving stack does not trade answer quality for throughput under load. Therefore, all RAG and LongBench leaderboards report absolute quality alongside SLO-goodput, and applications may add their own Qabs threshold. The synthetic quality guardrail catches speed-for-quality regressions, but full task quality for RAG and LongBench is reported separately and should not be reduced to goodput alone. We intentionally treat hardware, runtime, backend, containerization, and routing as part of the submitted serving system. This matches production deployment, but it means our pilot results do not isolate causal hardware effects. A controlled accelerator study would require matched backend maturity, identical router behavior, identical container/runtime layers, and symmetric multi-node availability across vendors.

References

- Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, and 1 others. 2025. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664.
- Yushi Bai and 1 others. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.
- Amanda Bertsch, Maor Ivgi, Emily Xiao, Uri Alon, Jonathan Berant, Matthew R Gormley, and Graham Neubig. 2025. In-context learning with long-context models: An in-depth exploration. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12119–12149.
- Longze Chen, Ziqiang Liu, Wanwei He, Yinhe Zheng, Hao Sun, Yunshui Li, Run Luo, and Min Yang. 2024. Long context is not long at all: A prospector of long-dependency data for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8222–8234.
- Zina Chkirbene, Ridha Hamila, Ala Gouisseem, and Unal Devrim. 2024. Large language models (llm) in industry: A survey of applications, challenges, and trends. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pages 229–234. IEEE.
- Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. *RAGAs: Automated evaluation of retrieval augmented generation*. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Krizan, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. *RULER: What’s the real context size of your long-context language models?* Preprint, arXiv:2404.06654.
- Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024a. Llm inference serving: Survey of recent advances and opportunities. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. IEEE.
- Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024b. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*.
- Percy Liang and 1 others. 2023. *Holistic evaluation of language models*. *Transactions on Machine Learning Research*.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.
- MLCommons. 2025. MLPerf Inference Datacenter Benchmark. <https://mlcommons.org/benchmarks/inference-datacenter/>. Accessed 2026.
- NVIDIA. 2025. LLM Benchmarking Metrics. <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html>. Accessed 2026.
- Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktaschel, and Sebastian Riedel. 2021. *KILT: a benchmark for knowledge intensive language tasks*. In *Proceedings of NAACL-HLT*, pages 2523–2544.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2023. [ARES: An automated evaluation framework for retrieval-augmented generation systems](#). *Preprint*, arXiv:2311.09476.

Nandan Thakur, Nils Reimers, Andreas Rucklé, Abhishek Srivastava, and Iryna Gurevych. 2021. [BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models](#). In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.

Maxwell Twelftree, David Lemphers, An-chi He, and Yue Yang. 2026. Not every sync is safe: Calibrated diloco scheduling for shared ai infrastructure. *arXiv preprint arXiv:2607.02544*.

Vaishali Vinay. 2025. Failure modes in llm systems: A system-level taxonomy for reliable ai applications. *arXiv preprint arXiv:2511.19933*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

A Reproducibility

See <https://github.com/Maincode-Research/opendc-infer-lc> for all implementation details.

B Reported Metrics

OpenDC-Infer-LC reports a single *primary* metric used for ranking and a broad set of *secondary* metrics for diagnosis. All percentiles are computed over the timed measurement window after warmup requests are excluded.

B.1 Latency Definitions

TTFT is the time to the first non-empty generated token. End-to-end (E2E) latency is measured from request submission to a clean close of the response stream. Time per output token (TPOT) for request i is

$$\text{TPOT}_i = \frac{t_i^{\text{end}} - t_i^{\text{first}}}{\max(y_i - 1, 1)}, \quad (1)$$

where t_i^{first} and t_i^{end} are the first-token and completion timestamps.

Table 5: OpenDC-Infer-LC workload definitions. LC-* are synthetic RULER-style needle-in-a-haystack; LongBench is the 21-task LongBench v1 suite; RAG-* perform real BM25 top- k retrieval over a shared 66 635-passage HotpotQA corpus, packed to the target budget (the same questions answered over progressively longer retrieved contexts).

Workload	Input Tokens	Max Output Tokens	Target Scenario
LC-8K	8K	512	Standard long-context (synthetic)
LC-32K	32K	512	Enterprise document QA (synthetic)
LC-128K	128K	1024	Long-document and codebase QA (synthetic)
LC-Cache	32K shared prefix + multi-turn queries	256 per turn	Agentic and repeated-prefix RAG
LongBench	$\leq 32K$ (21 real tasks)	32–512 (per task)	Real long-context QA / summarization / code
RAG-8K	8K (≈ 58 retrieved passages)	64	Real top- k retrieval, short context
RAG-32K	32K (≈ 223 retrieved passages)	64	Real top- k retrieval, enterprise QA
RAG-128K	128K (≈ 866 retrieved passages)	64	Real top- k retrieval, long context

C Benchmark Protocol and Submission Details

This appendix gives the full specification deferred from the main-body methodology: the synthetic-corpus generator, the endpoint measurement contract, the LC-Cache state taxonomy, the reference-harness run procedure, and the leaderboard/submission rules.

C.1 Corpus Generation

The long-context haystack is **procedurally generated from a seeded random number generator**, never sourced from public documents. Each prompt follows a RULER-style needle-in-a-haystack template: an instruction header, a synthetic haystack built from a seeded sentence generator over fixed adjective/noun/verb/ connective vocabularies, one or more embedded *needles*, and a question footer. A needle is a distinctive “magic number” statement (“*One of the special magic numbers for <key> is: <value>.*”) with key drawn from a fixed keyword set and value a random 7-digit integer, giving an unambiguous exact-match target for the quality guardrail. We use **single-key** (one needle) and **multi-key** (one queried needle plus decoy needles with different keys at evenly spread depths) variants. A length-control step re-tokenizes the decoded prompt with the declared tokenizer and truncates/pads the filler so the total hits the target token count *exactly* (≤ 5 -token drift, recorded per sample). Because token boundaries are tokenizer-specific, the frozen sets are generated *per tokenizer* (one tree per model) and pinned by a dataset_version_hash. Synthetic generation is chosen for three reasons: metric integrity (a repeated real passage is highly compressible and would inflate prefix-cache hit rates, distorting the cache metrics the benchmark measures), license

cleanliness, and bit-for-bit reproducibility.

For RAG, we build the workloads from the on-line distractor split, which contains 7,405 questions. Each question is paired with ten candidate paragraphs: two gold supporting paragraphs and eight distractors. The dataset also provides gold supporting-fact labels and a gold answer, which allow retrieval recall and answer quality to be measured independently.

A naive RAG construction would retrieve only from the ten paragraphs attached to each question, making the task short-context and largely trivial. Instead, we pool all distractor-split paragraphs across the 7,405 questions and deduplicate them by (title, text), yielding a shared corpus of 66,635 passages. All retrieval is performed against this full corpus, so each query searches a large haystack rather than its local candidate set. Retrieval uses a pure-Python Okapi BM25 implementation with an inverted index over the shared corpus. The retriever is deterministic, offline, and does not use a neural model or GPU. This makes the retrieval stage reproducible across hardware platforms and serving backends. For each query, deterministic BM25 ranks the shared 66,635-passage corpus. We greedily pack top-ranked passages to 8K, 32K, and 128K token budgets and record whether the gold supporting passages survive within the packed context. We construct three frozen splits: RAG-8K, RAG-32K, and RAG-128K. These correspond to progressively larger retrieved contexts, aveRAG-ing roughly 58, 223, and 866 passages per prompt, respectively. Each record contains the final RAG prompt (instruction, numbered passages, and question), the gold answer, and retrieval metadata computed at build time, including recall@1, recall@5, recall@10, recall@K, MRR, and whether all gold passages survived within the packed context bud-

get.

We score RAG along two independent axes. Retrieval quality is measured deterministically from the frozen build using recall@K and MRR. Answer quality is measured from model outputs using F1 and exact match against the gold answer. We also stratify answer quality by the gold-in-context flag, which separates retriever failures from generator failures: if the gold passages are absent from the packed context, the model may fail because retrieval missed the evidence; if the gold passages are present, failures more directly reflect generation or reasoning errors. This construction exposes the long-context RAG tradeoff: larger budgets admit more retrieved evidence, increasing recall@K (90.9%, 94.4%, and 96.9% for 8K, 32K, and 128K in our build) and allowing answer quality to improve when the serving system can sustain the longer contexts. All RAG splits are frozen per model tokenizer and evaluated through the same OpenDC-Infer-LC harness as the LC-8K, LC-32K, LC-128K, and LC-Cache workloads. Thus, synthetic long-context, LongBench-style, and retrieval-centric RAG workloads are evaluated under matched serving controls, SLO profiles, sampling settings, and context lengths.

C.2 Secondary Metrics (Full Set)

Beyond SLO-qualified goodput and scaling efficiency (Eq. 3.2.1), the harness reports the following;

- **Latency:** TTFT, TPOT, and E2E at p50/p95/p99.
- **Throughput:** raw output tok/s, total tok/s, and requests/s.
- **Capacity:** max concurrent users and max request rate under the SLO.
- **Reliability:** success, timeout, server-error, cancellation, and OOM rates.
- **Cache behavior:** prefix-cache hit rate, KV-cache usage, placement, eviction policy, and cache-reuse speedup.
- **Transport:** declared/observed transport, fallback count, and cross-node communication evidence.
- **Storage:** model/prompt/cache/log paths, read/write bandwidth, and whether each path is in the timed request path.
- **Hardware efficiency:** output tok/s per accelerator and, when available, output tokens per joule.
- **Run stability:** run-to-run variance, p99 drift, thermal-throttling indicators, telemetry gaps, and post-saturation behavior.

C.3 Endpoint Measurement Contract

The reference harness drives an OpenAI-compatible streaming endpoint (or equivalent). Because endpoints differ in streaming behavior, we fix:

- **Request start:** immediately before the client sends the request.
- **TTFT:** start until the first *non-empty* generated token; metadata-only or empty chunks do not count.
- **TPOT:** $TPOT_i = (t_i^{\text{end}} - t_i^{\text{first}}) / \max(y_i - 1, 1)$, with y_i output tokens and $t_i^{\text{first}}, t_i^{\text{end}}$ the first-token and completion timestamps.
- **E2E:** start until the final completion event is parsed and the stream closes cleanly.
- **Token accounting:** prompts/outputs re-tokenized with the declared tokenizer unless a validated server-side contract is provided.
- **Timeouts/cancellations:** timed-out, cancelled, malformed, or incomplete requests are unsuccessful and count against the success rate.
- **Retries:** disabled by default; if enabled for a diagnostic run, reported separately.

The submission package includes a loadgen_config.yaml specifying load-generator hosts, CPU count, NICs, network path, client processes, connection settings, TLS mode, tokenizer mode, timeouts, retry policy, and clock source. The load generator must not be the bottleneck at the reported peak: submissions report host count, CPU/NIC utilization, client process count, connection/timeout settings, tokenizer mode, and clock source, and a run in which the generator saturates CPU, network, file I/O, TLS, connection pools, or tokenization is invalid for ranking.

C.4 LC-Cache State Rules

Because cache state strongly affects results, cache modes are reported separately:

- **Cold cache:** no prefix/KV/semantic/document/storage cache populated before the timed window.
- **Benchmark-warmed cache:** reuse arises only from requests issued during the timed window.

- **Prefilled cache:** the shared prefix / reusable state is loaded before the timed window.
- **Persistent session cache:** state persists across turns within one synthetic conversation.
- **Cross-run cache:** state survives across independent runs—*prohibited* for ranked submissions unless disclosed in a separate non-default category.

Each LC-Cache submission discloses whether prefix caching, paged KV cache, KV offload, semantic cache, document cache, or storage-backed cache is enabled, and reports cache key granularity, capacity, eviction policy, observed hit rate, reset procedure, and memory location (HBM, host DRAM, local NVMe, shared filesystem, or remote object storage). LC-Cache leaderboards are reported separately for cold-, benchmark-warmed-, and prefilled-cache modes.

C.5 Reference Harness Procedure

The reference harness handles workload generation, request scheduling, streaming-response measurement, token accounting, SLO validation, quality checking, metadata collection, and report generation. It is backend-neutral and talks to any endpoint satisfying the measurement contract. A run proceeds:

1. Select workload, model tier, SLO profile, cache mode, and load-generation mode.
2. Verify the system passed the data-center qualification phase.
3. Warm up the serving system under a fixed warmup policy (including a priming request that absorbs first-request kernel/graph compilation).
4. Run the timed measurement window at a fixed offered load.
5. Record per-request metrics, streaming timestamps, system metrics, transport logs, and load-generator metrics.
6. Compute latency, reliability, raw throughput, SLO-qualification status, and SLO-qualified goodput.
7. Run the quality guardrail on the validation subset.

Workload	TTFT95	TPOT95	E2E95	Success
LC-8K	<3s	<0.12s/tok	<60s	>99%
LC-32K	<3s	<0.12s/tok	<60s	>99%
LC-128K	<10s	<0.20s/tok	<180s	>98%
LC-Cache	<3s	<0.12s/tok	<60s	>99%
RAG-8K	<3s	<0.12s/tok	<60s	>99%
RAG-32K	<10s	<0.30s/tok	<180s	>98%
RAG-64K [†]	<10s	<0.30s/tok	<180s	>98%
RAG-128K	<10s	<0.30s/tok	<180s	>98%
LongBench	<10s	<0.20s/tok	<180s	>98%

Table 6: Workload-specific SLO thresholds used for all reported results. All SLO checks use unrounded measurements; table values are rounded only for display. [†]RAG-64K is diagnostic-only and is not part of the v1.0 main leaderboard.

8. Emit the standardized submission package and report.

C.6 Leaderboard and Submission

OpenDC-Infer-LC reports multiple leaderboards rather than one aggregate score. The main leaderboard ranks by SLO-qualified goodput on LC-32K, our primary enterprise long-context document-QA setting; RAG-32K is reported as the primary retrieved-context enterprise QA setting.; additional leaderboards cover LC-8K interactive capacity, LC-128K heavy long-context capacity, LC-Cache prefix-reuse efficiency, and hardware efficiency. A submission is ranked only if: the run is SLO-qualified; the quality guardrail is satisfied; the system qualification phase passes; required metadata and raw artifacts are provided; and the cache-state, endpoint-measurement, and timed-window rules are followed. Submissions lacking complete qualification evidence may be listed as preliminary or partially qualified but are not ranked against fully qualified ones.

D Data-Center Benchmark Setting

OpenDC-Infer-LC evaluates a complete data-center serving path rather than an isolated model endpoint. A benchmarked system includes the selected compute nodes and accelerators, scheduler or orchestration layer, serving container and runtime, accelerator communication fabric, model and cache storage paths, telemetry stack, endpoint implementation, and load-generation clients. This boundary is necessary because long-context inference is often limited by system behavior outside the model kernel, including prefill scheduling, KV-cache placement, fabric fallback, storage staging, container limits, and load-generator saturation. Before timed runs,

each submission completes a qualification phase. The submitted artifact records the node allocation, accelerator count and visibility, host image, kernel, driver/runtime versions, container image digest, scheduler configuration, and serving launch command. For multi-node runs, submissions declare the intended accelerator transport and provide evidence of the observed transport from inside the benchmark process. If RDMA, RoCE, InfiniBand, or another vendor fabric is claimed, logs must distinguish fabric use from TCP or host-memory fallback. The benchmark also separates setup from timed execution. Submissions state whether model loading, tokenizer preprocessing, prompt-corpus staging, cache initialization, and image or package downloads occur before or during the measurement window. Model files, prompt corpora, and generated workload artifacts are identified by checksum or immutable revision. Storage and cache placement are reported for model weights, request corpora, prefix or KV cache, cache spill, logs, telemetry, and scratch space. Finally, the load generator is part of the measurement contract. Submissions report load-generator host count, CPU and NIC capacity, client process count, connection settings, tokenizer mode, retry policy, timeout policy, and clock source. A ranked result is valid only if the load generator has headroom at the reported peak or is included in the disclosed system boundary.

E Why SLO-Qualified Goodput

Raw token throughput is easy to inflate and easy to misread. The four measured cases below show, one failure mode at a time, why OpenDC-Infer-LC ranks on *SLO-qualified, quality-gated goodput* rather than on raw tokens/s. Each is taken directly from the runs in this paper.

(1) Higher concurrency raises raw throughput but breaks the latency SLO. Pushing offered concurrency from $C=8$ to $C=16$ on `1c_8k` (SGLang-35B) increases raw output throughput by 61%, but per-token latency crosses the interactive cap, so the point is rejected and contributes *nothing* to goodput.

C	raw out-tok/s	TPOT ₉₅ (s)	SLO (≤ 0.12)
8	83.2	0.103	✓
16	133.9	0.124	×

A throughput-only metric would report 134 tok/s; the SLO-honest answer caps the deliverable rate at $C=8$ (79.0 tok/s of goodput).

(2) A cache-heavy config looks fast but fails quality, so goodput is zero. On the multi-prefix cache workload `1c_cache_mp` (SGLang, round-robin), prefix reuse drives an enormous *total* token rate and the latency SLO is met—yet every answer fails the quality gate, so quality-gated goodput is 0.

C	raw out-tok/s	total tok/s	quality	goodput
8	102.1	738,266	0.00	0.0

The guardrail does exactly what it should: a fast-but-incorrect configuration earns zero deliverable goodput despite a headline 7.4×10^5 tok/s.

(3) A 128K config produces throughput but no SLO-qualified point. Long-context configurations ingest the prompt and emit tokens, but if even the single-request point ($C=1$) misses the decode-latency cap, there is no qualifying point and goodput is undefined (vLLM-35B):

config	total tok/s	TPOT ₉₅	SLO	goodput
RAG_128k	25,236	0.442 (> 0.30)	×	<i>none</i>
1c_128k	22,938	0.271 (> 0.20)	×	<i>none</i>

The same lengths *are* serviceable on a faster decode path—SGLang-35B RAG_128k qualifies at $C=1$ (0.93 tok/s, TPOT 0.186)—so the “none” is a property of the system-under-test, not of the workload.

(4) A routing policy preserves cache locality but collapses cluster goodput. On `lc_cache_mp` at $N=8$ nodes, prefix-hash routing pins each prefix to one replica (maximizing KV-cache reuse) but leaves cluster goodput at the single-node level, an $\approx 6.8\times$ collapse versus load-balanced routing (see Table 3):

strategy ($N=8$)	goodput (tok/s)	efficiency
round-robin	50.4	0.85
prefix-hash	7.4	0.12

Finding. Across all four cases the headline throughput number is either unearned or undeliverable: it is paid for by latency-SLO violations (1), by wrong answers (2), by length the system cannot serve within SLO (3), or by stranding 7/8 of the cluster behind cache affinity (4). SLO-qualified, quality-gated goodput collapses these four orthogonal failure modes into a single number that reflects what a user actually receives—which is why it, and not raw tokens/s, is the ranking metric.

Finding. For the reported pilot, we verified that the load generator was not the bottleneck at the ranked load points. Client CPU, NIC, tokenizer, file-I/O, TLS, and connection-pool utilization stayed below saturation; all requests in the timed window were labeled as successful, timed out, cancelled, malformed, or server-error; and representative repeated runs showed stable p95 latency and goodput. Runs failing these checks are excluded from ranking.

F LongBench Construction

F.1 Source and tasks

We use **LongBench v1** (4750 examples across 21 tasks): single- and multi-document QA, summarization, few-shot learning, synthetic retrieval/counting, and code completion. We take the raw task files plus the official `dataset2prompt`, `dataset2maxlen`, and metric definitions from the LongBench release, so prompts and scoring match the public leaderboard exactly.

F.2 Prompt templating and truncation

Each example is rendered with its task’s official prompt template, then **middle-truncated** to a token budget—retain the head and tail, drop the middle—using the served model’s tokenizer. We set the budget to **32,768** tokens: measured over the suite, the length distribution has $p_{50} \approx 6.7\text{K}$ and $p_{99} \approx 32\text{K}$ (words), so only $\approx 0.9\%$ of examples are truncated

at all—i.e. the canonical “LongBench-32K” setting rather than an arbitrary cap (Table 8). Per-task generation budgets come from `dataset2maxlen` (32–512 tokens).

F.3 Per-tokenizer builds and scoring

As with RAG, splits are frozen per model tokenizer so token budgets are exact. Scoring uses the official LongBench metrics ported verbatim—qa-F1, ROUGE-L, classification accuracy, retrieval/count exactness, and code edit-similarity, dispatched per task and macro-averaged—making the reported macro score leaderboard-comparable.

G Additional Benchmark Validation and Analysis

G.1 Relation to Existing Goodput Metrics and System Benchmarks

We do not claim that goodput or system-level benchmarking is new in isolation. DistServe uses SLO-constrained goodput primarily as an optimisation objective for a disaggregated prefill–decode architecture, whereas MLPerf provides broad and rigorous system-level inference evaluation across standardised models and scenarios. GenAI-Perf, by contrast, is a configurable endpoint-profiling tool. OpenDC-Infer-LC defines a different benchmark object and validity contract: *production-valid long-context serving capacity*.

OpenDC-Infer-LC applies goodput to the complete, backend- and hardware-neutral serving stack, and reports the highest offered load that simultaneously satisfies the declared TTFT, TPOT, end-to-end (E2E) latency, reliability, and task-quality requirements. The benchmark fixes reproducible 8K, 32K, and 128K synthetic and retrieved-context workloads; explicitly defines cold, benchmark-warmed, prefilled, and persistent-session cache states; includes prefix-reuse and multi-node routing workloads; and requires long-context-specific system evidence covering load-generator headroom, token accounting, KV-cache placement, transport, storage, and routing. These are not optional diagnostics: they determine whether a reported capacity is reproducible and deliverable.

When OpenDC-Infer-LC and an endpoint profiler are applied to the same endpoint, prompts, and request schedule, OpenDC-Infer-LC additionally tests whether the operating point is valid under a fixed long-context benchmark specification. For example, the B200 SGLang LC-128K throughput

Validation check	Reported evidence	Pass criterion
Load-generator headroom	CPU/NIC/tokenizer utilization	no client saturation
Token accounting	client/server token-count audit	mismatch <1%
Streaming timing	replayed synthetic endpoint	TTFT/TPOT error <1%
Trace completeness	request log audit	100% labeled requests
Run stability	3 repeated representative runs	goodput CV <5%
Clock source	host clock logs	monotonic, synchronized timestamps

Table 7: Reference-harness validation checks applied before ranking results.

	p_{50}	p_{90}	p_{95}	p_{99}	max
Length (words)	6672	16 104	22 672	31 935	44 221

Table 8: LongBench v1 input-length distribution (4,750 examples). At a 32,768-token budget only $\approx 0.9\%$ of examples require truncation.

increases from 1.283 to 1.867 output tokens/s at the next concurrency point, but the latter point is rejected because it violates the latency SLO. In another cache-heavy configuration, the endpoint reaches 102.1 output tokens/s while meeting the latency requirements, but receives zero OpenDC goodput because its answer quality is zero. A latency-only profiler would not reject this point unless a separate task-quality evaluation were implemented.

The novelty is therefore not merely the combination of existing metrics. Rather, OpenDC-Infer-LC operationalises the previously underspecified question—*how much correct long-context and RAG traffic can a complete cluster sustain?*—as a reproducible benchmark with explicit qualification rules. This shifts evaluation from peak endpoint throughput to production-valid cluster capacity.

The validity controls also reject several misleading headline results in our pilot: a 61% throughput increase obtained only after violating the latency SLO; approximately 7.4×10^5 total tokens/s with zero task quality; 128K configurations with no serviceable load point; and prefix-affinity routing that strands seven of eight nodes. The same harness, without code changes, also evaluates a second model family and exposes backend compatibility failures before throughput comparison is attempted.

G.2 Rationale and Sensitivity of Workload-Specific SLOs

Each default threshold is selected by mapping a workload to the service requirement that constrains how its output is consumed. We first divide the workloads into two service tiers. The interactive tier comprises LC-8K, LC-32K, LC-Cache, and

RAG-8K, for which a user is waiting for and reading the stream. The heavy-analysis tier comprises LC-128K, RAG-32K, RAG-128K, and LongBench, for which the user has committed to a longer wait for document or code analysis. Tiering SLOs by how application output is consumed is standard production practice. We then select the bound associated with the governing requirement for each tier. Table 9 summarises the resulting rationale.

Interactive bounds are therefore motivated by human response perception and reading speed; heavy-workload bounds are motivated by industry benchmark practice at the same service tier; and E2E bounds are motivated by explicit completion budgets and infrastructure timeouts. To test whether the reported backend rankings depend on the exact values, we perturb every latency bound simultaneously by $\pm 10\%$. All per-workload winners remain unchanged over all 729 evaluated grid points, and the rankings remain stable under perturbations as large as $\pm 25\%$.

G.3 Retriever Choice and Online Retrieval Overhead

The RAG workload separates two design choices: freezing the retrieved contexts and selecting a retriever to construct those contexts. Freezing is the load-bearing requirement for serving-system comparison because every backend and hardware stack must process byte-identical prompts. This requirement is compatible with sparse or neural retrievers. BM25 is used for the released benchmark build because it is deterministic and model-free, not because it is intended to represent the only production retrieval method.

To measure the effect of retriever choice, we replace BM25 with exact search using BGE-small-en-

Table 9: Rationale for the default workload-specific SLO thresholds.

Threshold	Deployment rationale
TTFT < 3 s—interactive	Keeps the first visible response within an interactive waiting window. The threshold lies within the established HCI response-time band and matches the 3,000 ms server-scenario TTFT used by MLPerf for its largest model profiles.
TTFT < 10 s—heavy	Uses the established 10-second HCI boundary at which users begin to lose focus or switch tasks, providing a defensible upper limit for 128K document- and code-analysis workloads.
TPOT < 0.12 s/token—interactive	Requires at least 8.3 tokens/s, corresponding to approximately 375 words/min under the conversion used in our analysis. This is above the approximately 300 words/min ceiling reported for normal adult silent reading, so the stream generally remains ahead of the reader. This comparison motivates the threshold; it is not presented as a formal production SLA.
TPOT < 0.20 s/token—heavy	Permits slower decoding for memory-intensive long-context workloads and is consistent with the 200 ms/token limit used in MLPerf conversational scenarios for models such as Llama 2 70B and Mixtral 8×7B.
TPOT < 0.30 s/token—long-context RAG	Long-context RAG workloads produce comparatively short answers, so their latency is dominated by retrieved-context prefill and TTFT rather than prolonged decoding. The E2E threshold continues to bound total response time.
E2E < 60 s / 180 s	Defines explicit completion budgets for interactive and heavy-analysis services. The 60-second limit prevents interactive requests from taking unreasonably long and is aligned with common cloud front-door idle timeouts; the 180-second limit accommodates the larger context and output budgets of heavy-analysis workloads.
Success > 99% / 98%	Provides per-load-point completion floors that expose saturation through timeouts, cancellations, server errors, and out-of-memory failures. These are measurement-window reliability gates, rather than monthly service-availability SLAs.

Table 10: Recall under BM25 and dense retrieval with fixed RAG token budgets.

Budget	k	BM25	Dense	Δ
RAG-8K	58	0.896	0.919	+0.023
RAG-32K	223	0.936	0.956	+0.021
RAG-128K	866	0.961	0.979	+0.018

v1.5 over the same 66,635-passage corpus, 7,405 queries, and fixed token budgets. Table 10 reports the result.

Dense retrieval improves recall by approximately two percentage points, with the largest change at the head of the ranking: MRR increases from 0.85 to 0.91. However, the retriever changes which passages are packed, not the number of packed tokens. Prefill cost, KV-cache footprint, and all serving-side measurements therefore remain fixed by the token budget and are identical under the two retrievers. Retriever choice changes only the quality diagnostics, which are stratified by whether the gold evidence is present in the retrieved context.

We additionally measure the latency that online retrieval would add to the timed path. Sparse BM25 retrieval takes 0.99 ms at p95, whereas dense retrieval, including query encoding and exact CPU search, takes 9.3 ms at p95. The latter is at most 3.4% of the measured RAG-8K TTFT p95 of 0.276

s. At this corpus size, moving retrieval into the measurement window therefore does not change the observed SLO outcome.

G.4 Cross-Family Backend Evaluation

To test whether the observed SGLang–vLLM pattern is specific to Qwen3, we run Llama-3.1-70B-Instruct, a dense GQA model without MoE, through the same harness without code changes. We use the pinned serving images unmodified, the same node and frozen benchmark splits, BF16, tensor parallelism of 8, eager execution, and a fresh server for each backend. The backend is the only experimental variable.

The LC-32K and LC-Cache results reproduce the direction of the Qwen3 findings, while LC-8K provides a counterexample in which vLLM is modestly faster. Both backends fail the LC-128K SLO for Llama-3.1-70B-Instruct. Thus, the additional model family supports the broader conclusion that backend rankings depend on workload regime, rather than establishing a universal winner.

G.5 Prefix-Cache Reuse and Cache-Diversity Sensitivity

We instrument the LC-Cache workload using vLLM prefix-cache counters scraped before and after each measurement window. Every point starts from a fresh server and is primed only with a trivial

Table 11: Backend comparison for Llama-3.1-70B-Instruct and the corresponding Qwen3 pattern reported in Table 2 of the main paper.

Workload	vLLM	SGLang	SGLang/vLLM	Qwen3 pattern (Table 2)
LC-8K	9.67	8.76	0.91×	SGLang 1.41–1.99×
LC-32K	1.70	2.55	1.50×	SGLang 1.80–2.60×
LC-128K	FAILSLO		—	SGLang only
LC-Cache	8.38 ($C^* = 1$)	24.12 ($C^* = 4$)	2.88×	SGLang 2.10–3.08×

Table 12: Measured prefix-cache reuse for LC-Cache.

Configuration	Hit rate	Goodput	SLO
Shipped, $C = 1$	99.80%	12.27	Pass
Shipped, $C = 8$	99.93%	47.83	Pass
Caching disabled, $C = 8$	—	1.96	Fail

Table 13: Sensitivity of LC-Cache to the number of distinct 32K prefixes.

Distinct prefixes	Hit rate	Goodput
1	99.93%	47.83
16	99.08%	38.64 (−19%)
64	95.77%	33.51 (−30%)

prompt. Table 12 reports the measured results for Qwen3-235B on eight MI355X GPUs.

Reuse is effectively complete in the shipped single-prefix setting, and approximately 96% of the LC-Cache result is attributable to caching: disabling the cache reduces goodput by a factor of 24. The $C = 8$ point also reproduces the value reported in Table 2 of the main paper, 47.83 versus the published 47.5.

The shipped workload uses `n_prefixes=1`: all 500 turns share one byte-identical 32K prefix. This setting represents the upper end of the prefix-reuse curve. To measure sensitivity to cache diversity, we hold the number of turns and offered tokens fixed while increasing the number of distinct prefixes. Table 13 shows that goodput remains substantial but declines as reuse becomes less concentrated.

H Workload-specific SLO thresholds

I Detailed Experiments Results

Table 14: Quality-preservation gate for MI355X BF16 runs with detailed per-load quality traces. For each model/backend/workload, C^* is the highest point that satisfies both the latency/reliability SLO and the quality-preservation gate. Rows marked failSLO have no ranked load point.

Model	Backend	Workload	Q_{ref}	Q_{abs}	Q_{rel}	Quality	Ranked?
235B-A22B	vLLM	LC-8K	1.00	1.00	1.00	Pass	Yes
235B-A22B	vLLM	LC-32K	1.00	1.00	1.00	Pass	Yes
235B-A22B	vLLM	LC-128K	1.00	—	—	—	failSLO
235B-A22B	vLLM	LC-Cache	1.00	1.00	1.00	Pass	Yes
235B-A22B	SGLang	LC-8K	1.00	1.00	1.00	Pass	Yes
235B-A22B	SGLang	LC-32K	1.00	1.00	1.00	Pass	Yes
235B-A22B	SGLang	LC-128K	1.00	1.00	1.00	Pass	Yes
235B-A22B	SGLang	LC-Cache	1.00	1.00	1.00	Pass	Yes
235B-A22B	vLLM	RAG-8K	0.60	0.60	1.00	Pass	Yes
235B-A22B	vLLM	RAG-32K	0.84	0.84	1.00	Pass	Yes
235B-A22B	vLLM	RAG-128K	0.75	—	—	—	failSLO
235B-A22B	SGLang	RAG-8K	0.78	0.78	1.00	Pass	Yes
235B-A22B	SGLang	RAG-32K	0.67	0.67	1.00	Pass	Yes
235B-A22B	SGLang	RAG-128K	0.62	0.62	1.00	Pass	Yes
35B-A3B	vLLM	LC-8K	1.00	1.00	1.00	Pass	Yes
35B-A3B	vLLM	LC-32K	1.00	1.00	1.00	Pass	Yes
35B-A3B	vLLM	LC-128K	1.00	—	—	—	failSLO
35B-A3B	vLLM	LC-Cache	1.00	1.00	1.00	Pass	Yes
35B-A3B	SGLang	LC-8K	0.93	0.94	1.01	Pass	Yes
35B-A3B	SGLang	LC-32K	0.96	0.96	1.00	Pass	Yes
35B-A3B	SGLang	LC-128K	0.93	0.93	1.00	Pass	Yes
35B-A3B	SGLang	LC-Cache	1.00	1.00	1.00	Pass	Yes
35B-A3B	vLLM	RAG-8K	0.69	0.70	1.01	Pass	Yes
35B-A3B	vLLM	RAG-32K	0.72	0.72	1.00	Pass	Yes
35B-A3B	vLLM	RAG-128K	0.83	—	—	—	failSLO
35B-A3B	SGLang	RAG-8K	0.69	0.70	1.01	Pass	Yes
35B-A3B	SGLang	RAG-32K	0.75	0.75	1.00	Pass	Yes
35B-A3B	SGLang	RAG-128K	0.70	0.70	1.00	Pass	Yes

Table 15: Single-node 235B results matrix. Q=SLO-qualified. “Eager” means HIP graphs disabled

config	workload	lvl	Q	goodput	TTFT ₉₅	TPOT ₉₅	E2E ₉₅	succ	qual
vLLM BF16 (eager)	lc_128k	1	×	0.233	19.7	1.04	26	1	1
vLLM BF16 (eager)	lc_32k	1	✓	3.62	1.74	0.035	1.95	1	1
vLLM BF16 (eager)	lc_32k	2	×	3.27	4.64	0.436	5.08	1	1
vLLM BF16 (eager)	lc_8k	1	✓	14.7	0.27	0.0364	0.485	1	1
vLLM BF16 (eager)	lc_8k	2	✓	22.2	0.535	0.0738	0.744	1	1
vLLM BF16 (eager)	lc_8k	4	×	33.6	0.807	0.153	1.23	1	1
vLLM BF16 (eager)	lc_cache	1	✓	12.1	0.349	0.0354	0.558	1	1
vLLM BF16 (eager)	lc_cache	2	✓	16.3	0.617	0.0741	0.836	1	1
vLLM BF16 (eager)	lc_cache	4	✓	32.7	0.625	0.0735	0.878	1	1
vLLM BF16 (eager)	lc_cache	8	✓	47.5	0.914	0.12	1.49	1	1
vLLM BF16 (eager)	lc_cache	16	×	59.5	1.62	0.139	2.21	1	1
SGLang BF16 (eager)	lc_128k	1	✓	0.933	6.95	0.0145	7.04	1	1
SGLang BF16 (eager)	lc_128k	2	×	0.933	13.8	1.13	14.1	1	1
SGLang BF16 (eager)	lc_32k	1	✓	6.53	0.971	0.0131	1.05	1	1
SGLang BF16 (eager)	lc_32k	2	×	6.77	2.01	0.152	2.17	1	1
SGLang BF16 (eager)	lc_8k	1	✓	13.7	0.213	0.0128	0.292	1	1
SGLang BF16 (eager)	lc_8k	2	✓	19.4	0.226	0.0465	0.505	1	0.99
SGLang BF16 (eager)	lc_8k	4	✓	31.3	0.803	0.0848	0.922	1	1
SGLang BF16 (eager)	lc_8k	8	×	35.6	1.56	0.185	1.73	1	1
SGLang BF16 (eager)	lc_cache	1	✓	25.6	0.169	0.013	0.25	1	1
SGLang BF16 (eager)	lc_cache	2	✓	32.7	0.325	0.0314	0.461	1	0.99
SGLang BF16 (eager)	lc_cache	4	✓	46.5	0.447	0.0406	0.643	1	1
SGLang BF16 (eager)	lc_cache	8	✓	65.1	0.627	0.0754	1.01	1	1
SGLang BF16 (eager)	lc_cache	16	✓	99.6	1.1	0.109	1.51	1	1
SGLang BF16 (eager)	lc_cache	32	×	102	2.12	0.175	2.92	1	1
vLLM FP8 (dynamic)	lc_128k	1	×	0.233	21.2	1.05	27.4	1	1
vLLM FP8 (dynamic)	lc_32k	1	✓	2.92	2.1	0.0464	2.38	1	1
vLLM FP8 (dynamic)	lc_32k	2	×	2.33	5.13	0.498	5.73	1	1
vLLM FP8 (dynamic)	lc_8k	1	✓	11.2	0.369	0.0436	0.627	1	1
vLLM FP8 (dynamic)	lc_8k	2	✓	16.8	0.724	0.0961	0.989	1	1
vLLM FP8 (dynamic)	lc_8k	4	×	24.7	1.07	0.202	1.64	1	1
vLLM FP8 (dynamic)	lc_cache	1	✓	10.8	0.359	0.0467	0.633	1	1
vLLM FP8 (dynamic)	lc_cache	2	✓	14.9	0.647	0.083	0.941	1	1
vLLM FP8 (dynamic)	lc_cache	4	✓	29.9	0.655	0.0819	0.973	1	1
vLLM FP8 (dynamic)	lc_cache	8	×	44.5	0.976	0.135	1.55	1	1
vLLM MXFP4	lc_128k	1	×	0.233	19.2	1.04	25.5	1	1
vLLM MXFP4	lc_32k	1	✓	3.85	1.63	0.0309	1.81	1	1
vLLM MXFP4	lc_32k	2	×	3.27	4.44	0.418	4.72	1	1
vLLM MXFP4	lc_8k	1	✓	0.583	0.353	0.0291	0.526	1	1
vLLM MXFP4	lc_8k	2	✓	21.2	0.477	0.0655	0.659	1	1
vLLM MXFP4	lc_8k	4	×	30.2	0.726	0.162	1.26	1	1
vLLM MXFP4	lc_cache	1	✓	12.6	0.352	0.0304	0.53	1	1
vLLM MXFP4	lc_cache	2	✓	17.1	0.609	0.0693	0.788	1	1
vLLM MXFP4	lc_cache	4	✓	34.2	0.617	0.0695	0.828	1	1
vLLM MXFP4	lc_cache	8	✓	48.3	0.931	0.118	1.27	1	1
vLLM MXFP4	lc_cache	16	×	59.7	1.58	0.14	2.2	1	1
vLLM BF16 (no-eager/graphs)	lc_128k	1	×	0.233	19.7	1.04	25.9	1	1
vLLM BF16 (no-eager/graphs)	lc_32k	1	×	1.98	1.88	0.272	3.5	1	1
vLLM BF16 (no-eager/graphs)	lc_8k	1	✓	9.1	0.301	0.0791	0.771	1	1
vLLM BF16 (no-eager/graphs)	lc_8k	2	✓	15.9	0.559	0.111	1.03	1	1
vLLM BF16 (no-eager/graphs)	lc_8k	4	×	25	0.796	0.179	1.45	1	1
vLLM BF16 (no-eager/graphs)	lc_cache	1	×	3.15	0.516	0.271	2.14	1	1
vLLM BF16 cold-cache	lc_cache	1	✓	3.03	2.09	0.0352	2.3	1	1
vLLM BF16 cold-cache	lc_cache	2	×	2.33	5.4	0.441	5.85	1	1
vLLM BF16 open-loop	lc_32k	1	×	5.6	127	0.759	129	1	1

Workload	vLLM BF16 G _{SLO} (C*)	SGLang BF16 G _{SLO} (C*)	TRT-LLM BF16 G _{SLO} (C*)	Best
LC-8K	36.4 (C=4)	38.27 (C=4)	32.2 (C=4)	SGLang
LC-32K	7.233 (C=1)	9.10 (C=2)	6.65 (C=1)	SGLang
LC-128K	1.167 (C=1)	1.283 (C=1)	1.167 (C=1)	SGLang
LC-Cache	64.63 (C=16)	63.82 (C=8)	129.85 (C=16)	TRT-LLM

Table 16: Completed single-node B200 results for Qwen3-235B-A22B (8×B200, TP=8, BF16, 132K served context)

Table 17: Single-node B200 complete results matrix: every completed backend × workload × load point. Qwen3-235B-A22B, 8×B200, TP=8, BF16

config	workload	lvl	Q	goodput	TTFT ₉₅	TPOT ₉₅	E2E ₉₅	succ	qual
vLLM BF16	lc_8k	1	✓	27.3	0.1929	0.0092	0.2476	1	1
vLLM BF16	lc_8k	2	✓	31.97	0.3774	0.0372	0.438	1	1
vLLM BF16	lc_8k	4	✓	36.4	0.6906	0.0924	0.7607	1	1
vLLM BF16	lc_8k	8	×	38.27	1.203	0.2015	1.442	1	1
vLLM BF16	lc_32k	1	✓	7.233	0.8666	0.0100	0.9254	1	1
vLLM BF16	lc_32k	2	×	7.467	1.734	0.139	1.796	1	1
vLLM BF16	lc_128k	1	✓	1.167	5.585	0.0077	5.623	1	1
vLLM BF16	lc_128k	2	×	1.400	8.903	0.2723	10.53	1	1
vLLM BF16	lc_cache	1	✓	33.48	0.1162	0.0097	0.1739	1	1
vLLM BF16	lc_cache	2	✓	42.70	0.2218	0.0169	0.3041	1	1
vLLM BF16	lc_cache	4	✓	55.53	0.4228	0.0266	0.5813	1	1
vLLM BF16	lc_cache	8	✓	62.53	0.8416	0.0475	1.085	1	1
vLLM BF16	lc_cache	16	✓	64.63	1.656	0.0971	2.149	1	1
vLLM BF16	lc_cache	32	×	75.02	2.968	0.1933	3.893	1	1
SGLang BF16	lc_8k	1	✓	30.57	0.1710	0.0083	0.2202	1	1
SGLang BF16	lc_8k	2	✓	35.00	0.3266	0.0258	0.3930	1	1
SGLang BF16	lc_8k	4	✓	38.27	0.6365	0.0814	0.7481	1	1
SGLang BF16	lc_8k	8	×	39.20	1.262	0.1923	1.473	1	1
SGLang BF16	lc_32k	1	✓	8.400	0.7431	0.0102	0.8031	1	1
SGLang BF16	lc_32k	2	✓	9.100	1.443	0.1137	1.577	1	1
SGLang BF16	lc_32k	4	×	9.333	2.845	0.3598	3.131	1	1
SGLang BF16	lc_128k	1	✓	1.283	6.459	0.0102	6.520	1	1
SGLang BF16	lc_128k	2	×	1.867	10.12	0.8334	10.36	1	1
SGLang BF16	lc_cache	1	✓	30.45	0.1350	0.0097	0.1925	1	1
SGLang BF16	lc_cache	2	✓	36.87	0.2564	0.0172	0.3567	1	1
SGLang BF16	lc_cache	4	✓	53.32	0.3857	0.0450	0.5976	1	1
SGLang BF16	lc_cache	8	✓	63.82	0.5897	0.0728	1.026	1	1
SGLang BF16	lc_cache	16	×	74.78	1.211	0.1324	1.887	1	1
TRT-LLM BF16	lc_8k	1	✓	21.93	0.2199	0.0161	0.3148	1	1
TRT-LLM BF16	lc_8k	2	✓	31.50	0.3901	0.0485	0.5095	1	1
TRT-LLM BF16	lc_8k	4	✓	32.20	0.7299	0.1067	0.8721	1	1
TRT-LLM BF16	lc_8k	8	×	36.40	1.387	0.2173	1.570	1	1
TRT-LLM BF16	lc_32k	1	✓	6.650	0.9194	0.0201	1.035	1	1
TRT-LLM BF16	lc_32k	2	×	7.233	1.766	0.1594	1.917	1	1
TRT-LLM BF16	lc_128k	1	✓	1.167	5.677	0.0249	5.816	1	1
TRT-LLM BF16	lc_128k	2	×	1.167	11.08	0.9274	11.42	1	1
TRT-LLM BF16	lc_cache	1	✓	26.02	0.1407	0.0176	0.2446	1	1
TRT-LLM BF16	lc_cache	2	✓	40.02	0.2010	0.0264	0.3381	1	1
TRT-LLM BF16	lc_cache	4	✓	59.03	0.2847	0.0395	0.4990	1	1
TRT-LLM BF16	lc_cache	8	✓	77.82	0.5892	0.0676	0.8570	1	1
TRT-LLM BF16	lc_cache	16	✓	129.85	0.6987	0.0869	1.062	1	1
TRT-LLM BF16	lc_cache	32	×	136.20	1.324	0.1538	1.877	1	1

Table 18: RAG (BM25 retrieve-to-budget) results for Qwen3-235B-A22B-Instruct-2507 on 8×MI355X (BF16, TP8). C =offered concurrency; SLO=✓if the point meets all p95 thresholds.

Backend	Workload	C	SLO	Goodput	TTFT ₉₅	TPOT ₉₅	E2E ₉₅	Succ	Qual
vLLM	RAG_8k	1	✓	4.00	0.276	0.037	1.87	1.00	0.60
		2		6.32	0.556	0.155	2.42	1.00	0.63
vLLM	RAG_32k	1	✓	1.77	1.852	0.037	2.55	1.00	0.84
		2		1.28	4.455	0.761	14.26	1.00	0.50
vLLM	RAG_128k	1		0.12	17.704	1.333	20.37	1.00	0.75
SGLang	RAG_8k	1	✓	1.27	0.254	0.064	1.49	1.00	0.78
		2		3.97	0.380	0.147	3.01	1.00	0.64
SGLang	RAG_32k	1	✓	2.55	0.859	0.067	3.13	1.00	0.67
		2		2.80	1.886	0.823	5.00	1.00	0.62
SGLang	RAG_128k	1	✓	0.22	7.167	0.063	8.48	1.00	0.62
		2		0.52	13.500	2.201	27.94	1.00	0.75