ASTRA-Sim Extension for Prefill-Decode Disaggregated LLM Serving with Chunked Prefill

1st Jingbin Lin
ECE
University of California San Diego
La Jolla, California
jil535@ucsd.edu

2nd Wentao Chen
CSE
University of California San Diego
La Jolla, California
wec056@ucsd.edu

3rd Tianyi Zhang
ECE
University of California San Diego
La Jolla, California
tiz054@ucsd.edu

***Abstract*—Large language model (LLM) serving is shaped by two phases with different resource behavior. Prefill processes the input prompt with high compute intensity, while decode repeatedly generates output tokens and is sensitive to iteration overhead, memory bandwidth, and batching interference. Recent systems therefore either keep prefill and decode colocated while reducing interference through chunking and hybrid batching, or disaggregate the two phases onto separate resources. This paper extends ASTRA-sim into a calibrated simulator for studying that design space. The extension represents each serving request as a dependency graph over prefill compute, optional KV-cache handoff, and iterative decode compute; maps graph nodes to colocated or prefill/decode-specialized GPU roles; and reports time-related serving metrics such as time to first token, time per output token, end-to-end latency, throughput, and goodput. The simulator does not execute model logits or sampling, so it does not predict generated-answer quality. We calibrate unchunked and chunked colocated and prefill-decode disaggregated modes with measurements from an NVIDIA L40S GPU server and validate simulator replay against the same request traces. In the non-overloaded validation scope, the unchunked simulator passes the acceptance gates with TTFT MAPE of 11.15% for colocated serving and 10.67% for disaggregated serving, while TPOT MAPE is 4.65% and 1.43%, respectively. We then use the calibrated unchunked constants to explore the colocated-disaggregated crossover, GPU-role assignment, and workload pressure. The current calibrated claim is limited to two-GPU serving; larger worker-count and four-GPU studies are simulator extrapolations that guide future hardware measurements.**

***Index Terms*—ASTRA-sim, LLM serving, prefill-decode disaggregation, calibration, simulation, KV cache**

I. Introduction

LLM serving alternates between a compute-heavy prefill over the input prompt and an iterative, latency-sensitive decode that emits one output token at a time. Colocating both phases simplifies deployment but lets long prefills delay decodes; disaggregating them reduces phase interference but introduces KV-cache transfer and role-assignment decisions. DistServe and Mooncake make the disaggregated choice explicit [2], [3], while SARATHI, Sarathi-Serve, DeepSpeed-FastGen, and POD-Attention improve colocated serving with chunking, hybrid batching, or prefill-decode overlap [4]–[7].

This project asks whether ASTRA-sim can be extended and calibrated to reproduce those serving dynamics, then explore configurations that are expensive to test directly in hardware. ASTRA-sim 2.0 already models graph-based workloads, communication, memory movement, and heterogeneous/disaggregated systems [1]; this extension specializes those primitives for online inference by mapping each request to phase-labeled prefill/decode work plus an optional KV handoff. The target is time-related serving behavior rather than answer quality: the simulator consumes arrivals, token counts, and fitted service costs, not model logits, sampling decisions, or generated text.

The completed evaluation first focuses on unchunked serving. We calibrate colocated and disaggregated modes on a two-GPU L40S server, validate the simulator against measured request traces, and then sweep larger prefill/decode worker assignments with the calibrated constants held fixed. This scope matters. The two-GPU `P1/D1` disaggregated point is calibrated evidence. Four-GPU layouts, larger worker counts, and tensor-parallel sensitivity studies are extrapolated simulator results until matching measurements are collected. The paper also reports chunked-prefill two-GPU replay validation for 128-, 256-, and 512-token chunks, but it does not yet promote chunked design-space comparisons into the main extrapolated study.

This paper makes three contributions:

- It describes a serving-oriented ASTRA-sim extension that represents colocated and prefill–decode disaggregated inference as dependency graphs over compute, communication, and memory-movement primitives.
- It presents a calibration and validation workflow anchored to measured two-GPU server output, with explicit TTFT, TPOT, E2E, throughput, and goodput acceptance gates.
- It uses the calibrated unchunked model to study GPU-role assignment and workload pressure, while clearly separating calibrated findings from simulator extrapolations.

II. Background and Metrics

A. Background

LLM serving usually contains two phases with different resource behavior. In the prefill phase, the model processes all prompt tokens and builds the KV cache. This phase has high arithmetic intensity and can saturate GPU compute even with a small number of requests. In the decode phase, the model generates one token per request at each iteration. Decode is

often less compute-efficient, but it is latency-sensitive because every output token depends on the previous one.

A major line of work keeps prefill and decode colocated on the same GPU pool but changes scheduling to reduce interference. SARATHI observes that long and uneven prefills can create pipeline bubbles and delay ongoing decodes. It therefore splits prefills into chunks and constructs decode-maximal mixed batches, where decode requests piggyback on prefill chunks with relatively low incremental cost [4]. Sarathi-Serve extends this idea to online serving. It uses near-equal prefill chunks, stall-free scheduling, and a token-budgeted batching policy to control per-iteration latency while preserving high throughput [5]. DeepSpeed-FastGen follows a similar motivation with Dynamic SplitFuse: long prompts are split into smaller prompt chunks and then recomposed with generation work, together with continuous batching and blocked KV-cache management [6].

Another direction improves the execution of hybrid prefill-decode batches below the scheduler level. POD-Attention argues that scheduler-level mixing alone is not enough, because prefill attention is more compute-intensive while decode attention is more memory-bandwidth-sensitive. Its kernel-level design jointly executes prefill and decode attention so that the two phases can overlap more effectively on GPU resources [7].

A different design choice is prefill-decode disaggregation. Mooncake separates prefill and decode onto different nodes and makes KV-cache movement explicit. It uses KV-cache reuse, incremental prefill, and asynchronous streaming transfer of newly generated KV cache from prefill nodes to decode nodes. This design can reduce phase interference and save decode-side GPU memory, but it introduces handoff cost and additional scheduling decisions between the two phases [3]. Together, these systems define the main design space studied in this project: optimized colocated serving with chunked or fused prefill, and disaggregated serving with explicit KV-cache transfer.

Prior work has also studied simulation for LLM serving performance. Vidur models LLM inference operators with profiling and predictive modeling, and uses the simulator to estimate end-to-end serving metrics such as latency and throughput under different workloads and system configurations [8]. LLMServingSim takes a hardware-software co-simulation approach and models serving behavior at the iteration level, so that accelerator and system designs can be explored without repeatedly running full hardware experiments [9]. Its newer version further targets heterogeneous and disaggregated LLM serving infrastructure, including runtime effects from batching, routing, offloading, memory, and interconnect behavior [10]. These works show that simulation is a practical tool for LLM serving design-space exploration. Our work follows the same motivation, but focuses on building an ASTRA-sim-based model and calibrating it for colocated and prefill-decode disaggregated serving.

B. Metrics

We use common LLM serving metrics to evaluate both measured systems and simulator outputs. For a request r, let t_r^{arr} be the arrival time, $t_r^{(1)}$ be the time when the first output token is produced, t_r^{last} be the completion time, and n_r^{out} be the number of generated output tokens.

Time to first token measures how long the user waits before seeing the first generated token:

$$\mathrm{TTFT}(r) = t_r^{(1)} - t_r^{arr}. \tag{1}$$

TTFT mainly captures request admission, prefill execution, possible queueing, and, in disaggregated serving, KV-cache handoff before decoding can start.

Time per output token measures the average latency of the remaining decode steps after the first token:

$$\mathrm{TPOT}(r) = \frac{t_r^{last} - t_r^{(1)}}{n_r^{out} - 1}. \tag{2}$$

TPOT reflects decode efficiency, batching behavior, and interference from other work. It is especially important for long responses, where most of the user visible latency comes from iterative decoding.

End-to-end latency is the total time from request arrival to completion:

$$E2E(r) = t_r^{last} - t_r^{arr}. \tag{3}$$

We also report request throughput, measured as completed requests per second, and output-token throughput, measured as generated output tokens per second. Finally, goodput counts only the requests that satisfy a configured service-level objective. This is useful because a system can have high raw throughput while still violating latency requirements for many requests.

C. Prefill-Decode Serving

For a request r, prefill consumes the prompt tokens and creates the KV state needed by decode. Decode then produces n_r^{out} output tokens autoregressively. In a colocated design, prefill and decode execute on the same logical GPU pool or replica. In the calibrated baseline used in this paper, that pool is one L40S GPU, but the simulator treats colocation as a placement policy rather than requiring exactly one physical GPU. This avoids explicit inter-stage transfer, but prefill and decode contend for the same queues and kernels. In a prefill-decode disaggregated design, prefill work is assigned to a prefiller GPU pool and decode work to a decoder GPU pool. Disaggregation can remove phase interference, but it adds KV-cache handoff cost and may introduce separate prefill-side, transfer-side, and decode-side queues.

We use the standard serving metrics emphasized by prior LLM serving systems [2], [5]. We report end-to-end latency $t_r^{last} - t_r^{arr}$, request throughput, output-token throughput, and goodput. Goodput counts only requests that satisfy the configured service-level objective (SLO).

D. ASTRA-sim Substrate

ASTRA-sim already exposes much of the machinery needed for a serving simulator. Its workload layer is graph-based, and nodes can represent compute, communication, and memory behavior. Its system layer schedules streams through active queues and tracks resource activity. The serving extension keeps this substrate and changes the workload semantics: instead of treating an execution trace as a training loop, it emits a serving DAG per request and tags the nodes with phase, placement, and metric metadata.

III. Simulator Extension

A. Colocated Mode Extension

The colocated extension models the traditional way an LLM is served: a GPU owns the full request lifecycle, running prefill for the prompt and then running the autoregressive decode steps for the same request. The request configuration records the overall server GPU budget and the number of same-pool serving replicas. The topology creates one colocated GPU group per replica. When a request is admitted to a group, both its prefill and its decode steps remain on that same group; no prefill/decode role split is introduced. For a request r, the logical dependency chain is therefore

$$P_r \rightarrow D_{r,1} \rightarrow D_{r,2} \rightarrow \cdots \rightarrow D_{r,n_r^{out}}. \tag{4}$$

Arrivals first enter a global admission queue. The colocated runtime assigns each admitted request to one colocated group subject to the configured concurrency limit. The simulator may batch multiple requests within that group, but the batching remains local to the same colocated GPU resource: prompt batches charge prefill service, and decode batches charge one output-token step for each included request. This is still a same-GPU, same-pool execution model. The simulator records prefill and decode timestamps separately for calibration, but those timestamps are accounting boundaries rather than separate hardware roles. No explicit KV-transfer stage is inserted; once prefill completes, the KV state is treated as resident for decode on the colocated side.

B. PD Disaggregation Extension

The PD extension keeps the same request-level prefill and decode semantics, but splits the server budget into role-specific GPU pools. The configuration chooses how many GPUs serve as prefill workers and how many GPUs serve as decode workers. Optional tensor-parallel, pipeline-parallel, expert-parallel, and data-parallel-attention settings describe the cost-model structure of those groups, while the prefill/decode GPU counts determine the schedulable worker pools. These roles are fixed for the duration of a run. An idle decode GPU is not temporarily reassigned to prefill, and an idle prefiller is not temporarily reassigned to decode. This fixed-role policy matches the vLLM-style standard PD disaggregation mode used as the calibration target, where engines are provisioned as prefill producers or decode consumers. A dynamic-role policy would be a scheduler state-machine extension, but it is outside the calibrated scope of this paper. A PD request inserts an explicit KV-cache handoff between prefill and decode:

$$P_r \rightarrow K_r^{send} \rightarrow K_r^{recv} \rightarrow D_{r,1} \rightarrow \cdots \rightarrow D_{r,n_r^{out}}. \tag{5}$$

The prefill node is mapped to the prefiller set, the receive and decode nodes are mapped to the decoder set, and the send/receive pair accounts for the inter-stage KV transfer. The runtime therefore maintains separate prefill, transfer, and decode queues. After prefill finishes, the simulator either schedules an explicit KV transfer or, when transfer is disabled for a zero-cost fast path, marks the KV state as resident on the decode side. Decode then repeatedly schedules one output-token step until the request completes.

Fig. 1 shows this request-level PD execution path. The calibrated hardware point in this paper is the two-GPU `P1/D1` layout: one L40S GPU for prefill and one L40S GPU for decode. Larger prefill/decode counts use the same structural extension, but they are extrapolated simulator studies until matching measurements are collected.

```
procedure SimulatePDRequest(r)
wait until the arrival time of r
replica ← AssignReplica(r)
Enqueue(Q_P[replica], r)
while r is not finished do
    if an idle prefill worker exists then
        B_P ← BuildPrefillBatch(Q_P[replica])
        Run B_P for EstimatePrefillTime(B_P)
        for all request x in B_P do
            if x has remaining prefill tokens then
                Enqueue(Q_P[replica], x)
            else if transfer is enabled then
                Enqueue(Q_K, x)
            else
                MarkKVResidentOnDecode(x)
                Enqueue(Q_D[replica], x)
            end if
        end for
    end if
    if Q_K is nonempty then
        x ← Dequeue(Q_K)
        S_K ← EstimateKVBytes(x)
        Run transfer for α_net + β_net S_K
        MarkKVResidentOnDecode(x)
        Enqueue(Q_D[replica], x)
    end if
    if an idle decode worker exists then
        B_D ← BuildDecodeBatch(Q_D[replica])
        Mark first-token time for first decode step
        Run B_D for EstimateDecodeStepTime(B_D)
        Finish completed requests; requeue the rest
    end if
    Update TTFT, TPOT, E2E, throughput, and goodput metrics
end while
```

Fig. 1. Request-level pseudocode for prefill–decode disaggregated serving. Q_P, Q_K, and Q_D denote prefill, transfer, and decode queues.

C. Chunk Mode Extension

Chunk mode is implemented as a prefill-batching transformation rather than as a new serving phase. Each request state stores the remaining prompt tokens. When scheduler.chunked_prefill_size=0, the batch builder issues the entire remaining prompt as one prefill item. When the value is positive, each prefill item is capped at

$$q_r = \min(R_r^P, C), \tag{6}$$

where R_r^P is request r's remaining prompt tokens and C is the configured chunk size. The runtime subtracts q_r from R_r^P, increments the request's prefill-chunk counter, and requeues the request for another prefill batch until the full prompt has been consumed. The same batch builder still enforces the configured prefill token and request limits, so chunking changes the granularity of prefill work while preserving the existing queue and worker abstractions.

The activation path differs by parent mode. For colocated serving, the parser exposes a distinct colocated_chunked architecture and rejects it unless chunked_prefill_size¿0. The colocated runtime enables chunking only for that architecture. Each prefill chunk and every subsequent decode step remain on the same colocated GPU group; after the final prefill chunk, the request is marked decode-ready and no KV-transfer stage is inserted. Thus colocated chunk mode models the effect of breaking long prefills into shorter schedulable units, not a split between prefill and decode hardware.

For PD serving, the C++ architecture remains pd_disaggregated; a positive chunked_prefill_size turns on chunking inside the prefiller queues. Chunks run only on prefill worker groups. If prompt tokens remain after a chunk finishes, the request is pushed back to the PD prefill queue. Only after the final chunk does the request enter the transfer queue, or go directly to the decode queue when transfer is disabled. The implementation therefore performs one modeled KV handoff per request after complete prefill, not one handoff per prefill chunk.

The metrics path mirrors this state machine. Request CSVs and summaries report prefill_chunk_count, max_prefill_chunk_tokens, transfer_handoff_count, stage service times, and accumulated queue waits such as total_prefill_queue_wait_ns. Event and stage traces also expose the scheduled prefill-token counts, which the regression tests use to check splits such as $[2, 2, 1]$ for a five-token prompt with a two-token chunk size. These fields make chunked replay observable, but the model is still an ASTRA-sim queue-level abstraction: it does not model vLLM cache block IDs, prefix-cache hits, preemption, or per-layer asynchronous KV transfer. The present paper therefore treats chunked-prefill validation as diagnostic support, while the primary design-space claims remain the calibrated unchunked colocated and unchunked PD sweeps.

D. Service Model

The simulator is calibrated empirically, but the design-space results are easier to interpret with a latency decomposition that mirrors the runtime's stage accounting. For a colocated request,

$$T_r^{co} = W_r^{P,co} + C_r^{P,co} + \sum_{k=1}^{n_r^{out}} \left(W_{r,k}^{D,co} + C_{r,k}^{D,co}(\lambda) \right). \tag{7}$$

Here $W_r^{P,co}$ is colocated admission/prefill queue wait, $C_r^{P,co}$ is prefill service, $W_{r,k}^{D,co}$ is per-step decode queue wait, and $C_{r,k}^{D,co}(\lambda)$ is the configured decode-step service produced by the runtime's decode breakdown. In disaggregated mode,

$$T_r^{pd} = W_r^P + C_r^P + W_r^{KV} + C_r^{KV} + \sum_{k=1}^{n_r^{out}} \left(W_{r,k}^D + C_{r,k}^D(\lambda) \right). \tag{8}$$

Here W_r^P and C_r^P are prefill queue wait and service on the prefiller, W_r^{KV} is transfer-queue wait, C_r^{KV} is transfer service, $W_{r,k}^D$ is per-step decode queue wait, and $C_{r,k}^D(\lambda)$ is decode-step service on the decoder. If transfer is disabled or the KV payload is zero, $C_r^{KV} = 0$; otherwise,

$$C_r^{KV} = \alpha_{net} + \beta_{net} S_r^{KV}, \qquad S_r^{KV} = p_r \cdot m_{kv}, \tag{9}$$

where p_r is prompt length and m_{kv} is the KV footprint per prompt token.

The runtime models first-token visibility separately from decode completion. When first-token timing is modeled from decode start,

$$\mathrm{TTFT}(r) = t_r^{decode,start} + F_r(\lambda, i_r) - t_r^{arr}, \tag{10}$$

where $F_r(\lambda, i_r)$ is the calibrated first-token offset for request rank i_r. In the current PD fit, this term can come from an arrival-rank backpressure curve; in the calibrated colocated example, the offset is zero.

Disaggregation is beneficial when the resulting request latency is lower:

$$T_r^{pd} < T_r^{co}. \tag{11}$$

Colocated phase interference is therefore represented by scheduler-driven queueing and configured service terms such as decode batch efficiency or decode interference scaling, not by a separate runtime variable. For worker assignment, we also use a stage-balance proxy:

$$U_P = \frac{\lambda \mathbb{E}[C^P]}{G_P}, \qquad U_D = \frac{\lambda \mathbb{E}[n^{out}] \mathbb{E}[C^D(\lambda)]}{G_D}, \tag{12}$$

where λ is request rate and G_P, G_D are prefill and decode GPU counts. The best assignment should avoid making either stage the dominant bottleneck. The chosen prefill/decode worker counts affect queue availability and stage pressure; the current calibrated decode-step service curve is rate-dependent, not a measured multi-GPU decode-efficiency model.

IV. Calibration and Evaluation Method

A. Calibration Scope

The calibrated result package covers two unchunked modes: colocated serving and PD-disaggregated serving on one two-GPU NVIDIA L40S server. The measured colocated layout uses one same-pool serving GPU, while the measured PD layout allocates one GPU to prefill and one to decode. The simulator keeps this distinction explicit: colocated serving is a same-pool abstraction, whereas PD serving allocates prefill and decode workers separately.

Each replay trace combines ShareGPT-derived request shapes with Poisson arrivals. The trace records arrival offset, prompt length, requested output length, source row, request rate, and random seed, so the same time series can drive hardware measurement and simulator replay. Calibration first runs the trace on the target hardware mode, then fits mode-specific cost terms and replays the same trace identifiers so measured and simulated rows pair by mode and run id.

The fitted model includes effective prefill and decode terms for colocated serving, and prefill service, optional KV-transfer latency/bandwidth, a rate-dependent decode-step curve, and first-token backpressure for PD. Replay metrics retain per-stage queue, start, end, and service timestamps, allowing the fit to compare stage waiting and service rather than only request-level latency. The acceptance scope excludes the explicit 16 request/s overload trace: latency metrics must be within 15% MAPE, and throughput/goodput within 1% MAPE. Overload remains a diagnostic case with looser tolerance.

B. Extrapolation Scope

After validation, the calibrated constants are reused unchanged for simulator sweeps. The colocated-vs-PD crossover study compares four colocated replicas against `P1/D3`, `P2/D2`, and `P3/D1` four-GPU PD layouts over prompt and output length at 8 req/s. The compact scaling study varies worker counts and tensor-parallel degrees at one operating point. The four-GPU worker-assignment study then compares `P1/D3`, `P2/D2`, and `P3/D1` with tensor-parallel degree fixed at one. The pressure study sweeps request rate and prompt length for four-GPU assignments. The output-length study sweeps request rate and average generated tokens. These studies are useful for ranking candidate configurations, but only the `P1/D1` two-GPU point is calibrated against hardware. All PD sweeps use static prefill/decode role counts, so the reported winners rank fixed placements rather than adaptive policies that let idle GPUs change role during a run.

V. Results

A. Two-GPU Calibration Closure

Table I summarizes the validation gate. The unchunked colocated and PD-disaggregated replays both pass the non-overloaded acceptance scope. The colocated replay stays within the 15% latency threshold with 11.15% TTFT MAPE, 4.65% TPOT MAPE, and 4.86% E2E MAPE. The unchunked PD replay is tighter on decode-side and end-to-end behavior, with 10.67% TTFT MAPE, 1.43% TPOT MAPE, and 1.67% E2E MAPE. Goodput and request throughput are within 0.11% MAPE across both unchunked modes. The chunked rows aggregate fixed-size replay validation for 128-, 256-, and 512-token prefill chunks; they show that the chunked state machine is observable and replayable, but the four-GPU chunked design-space points below are still extrapolated.

The validation result does not claim cycle-level fidelity. It establishes a bounded replay model that preserves measured serving trends closely enough to rank candidate configurations and choose the next hardware experiments.

TABLE I
Validation MAPE on Two-GPU Measured Traces

Mode	Pairs	TTFT (%)	TPOT (%)	E2E (%)	Goodput (%)	Req. thpt (%)
C-unchk	16	11.15	4.65	4.86	0.00	0.00
PD-unchk	15	10.67	1.43	1.67	0.11	0.11
C-chk	63	6.21	5.73	5.71	0.30	0.00
PD-chk	61	4.30	3.65	3.73	0.00	0.00

Mean input tokens \ Mean output tokens	64	128	256	512
3072	C4 (+45)	P2/D2 (-11)	P2/D2 (-111)	P1/D3 (-440)
4096	P2/D2 (-18)	P2/D2 (-172)	P2/D2 (-435)	P2/D2 (-946)
6144	P2/D2 (-64)	P2/D2 (-440)	P2/D2 (-1,161)	P2/D2 (-2,477)
7168	P3/D1 (-173)	P3/D1 (-473)	P2/D2 (-1,412)	P2/D2 (-3,168)
10000	P3/D1 (-511)	P3/D1 (-1,379)	P3/D1 (-3,069)	P3/D1 (-5,588)

Fig. 2. Selected 4 req/s token-shape slice from Table A.1. Cells cover mean input tokens 3072–10000 and mean output tokens 64–512. Each cell shows the lowest-E2E global winner and the best-PD-minus-`C4` mean E2E margin in milliseconds. Negative margins mean the PD winner is faster than `C4`. All cells are extrapolated.

B. Colocated-PD Crossover

We first use the unchunked model to compare a four-GPU colocated data-parallel layout, `C4`, against three four-GPU PD splits: `P1/D3`, `P2/D2`, and `P3/D1`. Winners are selected by lowest mean E2E latency, because the simulator is being used to identify regimes for follow-up measurement rather than to enforce a production SLO. At 8 req/s, the first global PD transitions occur in three distinct workload regions: decode-heavy `P1/D3` first wins at 128 input tokens and 1024 output tokens, balanced `P2/D2` first wins at 3072 input tokens and 64 output tokens, and prefill-heavy `P3/D1` first wins at 4096 input tokens and 32 output tokens. Across the 70-cell 8 req/s matrix, `C4` still wins the majority of cells, while all three PD splits appear in the winner set. The appendix reports the expanded 4 req/s matrix, which adds the 10000-input and 2048-output corners and keeps the same lowest-E2E global-winner rule.

C. Compact Scaling Sweep

The compact PD sweep checks which scaling knobs currently affect the calibrated service model. It holds the workload at 8 rps, a 224-token mean prompt, and a 28-token mean output, then varies worker count and tensor-parallel degree. Tensor-parallel-only cases remain essentially equal to the calibrated `P1/D1` baseline: 187.81 ms TTFT, 33.52 ms TPOT, and 1106.21 ms E2E latency; the largest E2E change among TP-only rows is 0.00 ms. Worker replication changes queue

availability: balanced P2/D2 reduces mean E2E to 915.02 ms, balanced P4/D4 to 810.09 ms, and balanced P8/D8 to 766.60 ms. This is a directional simulator signal, so the following sections focus on worker placement and workload pressure rather than claiming measured multi-worker efficiency.

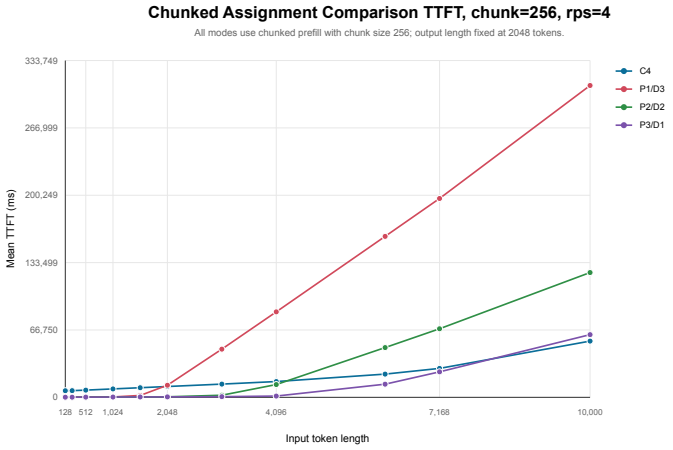

Fig. 3. Chunk-256 assignment comparison for mean TTFT at 4 req/s and 2048 output tokens. All modes use chunked prefill; all four-GPU points are simulator extrapolations.

D. GPU Role Assignment

The fixed four-GPU placement sweep holds tensor-parallel degree at one and uses the calibrated two-GPU P1/D1 row as its reference point. Adding workers improves TTFT and E2E because the explicit placement counts create additional schedulable prefill/decode groups. The best four-GPU assignment in this sweep is P1/D3, with mean TTFT of 148.28 ms and mean E2E latency of 851.34 ms, compared with 187.81 ms TTFT and 1106.21 ms E2E for the calibrated P1/D1 reference. The small TPOT spread is a limitation rather than a measured speedup: the current decode-step curve is calibrated by request rate and not by measured multi-worker decode efficiency. As a result, worker scaling mostly changes queueing and prefill-side pressure in this sweep.

E. Chunked-Prefill Sweep

The refreshed chunked experiment fixes the request rate at 4 req/s and the output length at 2048 tokens, then varies input length, chunk size, and four-GPU assignment. This experiment addresses the strongest colocated baseline suggested by chunked-prefill systems [4]–[7]: PD should be compared against chunked colocated serving, not only against unchunked colocation.

Figure 4 shows that chunk size remains a first-order control knob. For both C4 and P2/D2, the best chunk follows the prompt length up to 1024 tokens and then stays at 1024 for longer prompts. With that best chunk size, P2/D2 keeps TTFT below 1.5 s through 7168 input tokens and reaches 11.89 s at 10000 input tokens; the best C4 chunk reaches 15.77 s at the same 10000-token point. Figure 3 then fixes chunk size at 256 and compares role assignments. The TTFT winner moves from P1/D3 at 128–256 input tokens, to P2/D2 at 512–1024 tokens, to P3/D1 from 1536 through 7168 tokens, before C4 has the lowest TTFT at the 10000-token overload corner. E2E latency does not select the same winners: at chunk size 256, P1/D3 wins five short-prompt cells, P2/D2 wins three middle cells, and C4 wins the three longest-prompt cells. The chunked result therefore strengthens the main caution of the paper: optimized colocation can re-enter the winner set, and TTFT-optimal assignments are not always E2E-optimal assignments.

F. Workload Pressure

The unchunked four-GPU pressure view is drawn from the 4 req/s global crossover matrix, which varies mean input and output tokens across colocated DP and PD worker assignments. Figure 2 selects the high-pressure region from 3072 input tokens and 64 output tokens through 10000 input tokens and 512 output tokens. This slice exposes where the best lowest-E2E winner moves from colocated DP into balanced and prefill-heavy PD splits.

Figure 2 summarizes the workload-pressure result for the selected 4 req/s input/output region. In this slice, P2/D2 wins 12 cells, P3/D1 wins 6 cells, C4 wins 1 cell, and P1/D3 wins 1 cell. This is the stage-balance model in practice: prompt-heavy regimes prefer prefill capacity, while balanced input/output load keeps the split near P2/D2. The fixed-rate input/output crossover matrix is reported with the lowest-E2E global winner rule in Appendix A.

G. Output Length Sensitivity

The global token-shape crossover tables carry the main input/output crossover claim. The separate output-length sweep is kept as a sensitivity check while holding the mean prompt length at 512 tokens. At 8 req/s, the lowest-E2E winner among tested four-GPU assignments is P1/D3 for all four output-length cells. Request rates above 12 req/s are treated as overload diagnostics and are excluded from calibrated-comparison claims.

VI. Discussion

A. What the Simulator Can Claim

The strongest completed claim is calibration closure for two-GPU serving. The simulator reproduces non-overloaded TTFT, TPOT, E2E latency, throughput, and goodput within the configured gates for unchunked colocated and unchunked PD serving, and the averaged chunked-prefill replay rows in Table I show that fixed-size chunked execution can also be replayed with bounded error. That evidence supports controlled simulator exploration, especially for choosing which worker assignments and workload regimes deserve real four-GPU measurement. The claim is intentionally limited to serving-time behavior. Because ASTRA-sim does not execute the model or sampling loop, chunk size and role assignment results do not say whether a generated answer would be more accurate, fluent, or otherwise higher quality.

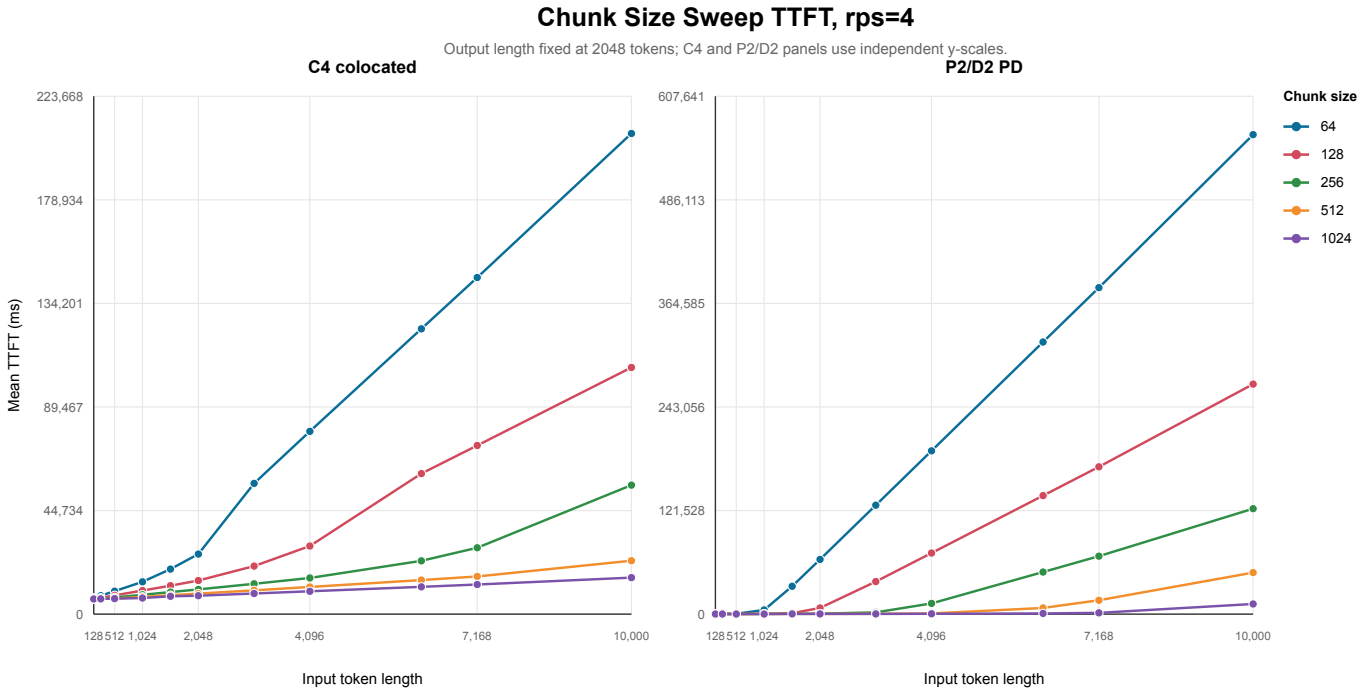

Fig. 4. Chunk-size sweep for mean TTFT at 4 req/s and 2048 output tokens. The `C4` and `P2/D2` panels use independent y-scales. All four-GPU points are simulator extrapolations.

The extrapolated studies should be read as sensitivity analysis. The unchunked crossover matrix and the chunked TTFT figures agree on one higher-level point: GPU count alone is not the right explanatory variable. A fixed GPU budget must be divided according to phase pressure and the objective being optimized. Decode-heavy, balanced, prefill-heavy, and colocated assignments all appear in the winner set once the workload shape, chunk size, and metric change. This matches the service model in (11) and (12): PD helps when reduced phase interference and queueing outweigh KV handoff and stage imbalance, but optimized chunked colocation can still be competitive.

B. Limitations

There are five important limitations. First, only the two-GPU colocated point, the two-GPU `P1/D1` disaggregated point, and fixed-size two-GPU chunked replay rows are calibrated. Four-GPU assignments, larger worker-count sweeps, and the chunked four-GPU figures are not deployment recommendations until real measurements are collected. Second, the PD scheduler uses fixed prefill and decode roles to stay aligned with the vLLM-style standard disaggregation mode used for calibration. It does not let an idle GPU temporarily borrow the other role. Such adaptive role switching is a natural extension of the scheduler state machine, but this paper does not calibrate or evaluate it. Third, the current decode-step model does not yet capture measured multi-worker decode efficiency, which is why TPOT changes only modestly in the worker-assignment sweep. Fourth, chunk mode is still a queue-level abstraction. It exposes prefill chunks, per-request handoff counts, and stage timing, but it does not model vLLM block management, prefix-cache hits, preemption, or per-layer asynchronous KV transfer. Those mechanisms matter for a final comparison against production-quality chunked serving. Fifth, all results are time-related serving metrics. The simulator does not run LLM inference, logits, decoding policies, or evaluation benchmarks, so it cannot predict answer quality or task accuracy.

VII. Conclusion

This work turns ASTRA-sim into a calibrated simulator for prefill–decode serving by expressing requests as dependency graphs over compute, communication, and memory movement, fitting the model to a two-GPU L40S server, and validating simulator replay against measured request traces. The validated simulator passes the non-overloaded acceptance gate for unchunked colocated, unchunked prefill–decode disaggregated, and fixed-size chunked replay modes. The extrapolated sweeps show that there is no single best four-GPU serving layout: unchunked crossovers move from colocated DP to decode-heavy, balanced, and prefill-heavy PD splits as token shape changes, while chunked-prefill experiments show that chunk size and the chosen metric can bring optimized colocation back into the winner set. The practical outcome is a measurement workflow rather than a deployment prescription: anchor ASTRA-sim with hardware traces, use simulation to identify break-even regimes and candidate layouts, and then return to hardware for the larger worker-count and production-quality chunked-prefill comparisons that remain outside the current calibrated claim.

REFERENCES

[1] ASTRA-sim Project, "ASTRA-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale," arXiv:2303.14006, 2023.

[2] Y. Zhong *et al.*, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *Proc. USENIX OSDI*, 2024.

[3] R. Qin *et al.*, "Mooncake: A KVCache-centric disaggregated architecture for LLM serving," arXiv:2407.00079, 2024.

[4] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "SARATHI: Efficient LLM inference by piggybacking decodes with chunked prefills," arXiv:2308.16369, 2023.

[5] A. Agrawal *et al.*, "Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve," in *Proc. USENIX OSDI*, 2024.

[6] C. Holmes *et al.*, "DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference," arXiv:2401.08671, 2024.

[7] A. Panwar *et al.*, "POD-Attention: Unlocking full prefill-decode overlap for faster LLM inference," arXiv:2410.18038, 2024.

[8] A. Agrawal, N. Kedia, J. Mohan, A. Panwar, N. Kwatra, B. S. Gulavani, R. Ramjee, and A. Tumanov, "Vidur: A large-scale simulation framework for LLM inference," in *Proc. MLSys*, 2024.

[9] J. Cho, M. Kim, H. Choi, G. Heo, and J. Park, "LLMServingSim: A HW/SW co-simulation infrastructure for LLM inference serving at scale," in *Proc. IEEE IISWC*, 2024.

[10] J. Cho, H. Choi, G. Heo, and J. Park, "LLMServingSim 2.0: A unified simulator for heterogeneous and disaggregated LLM serving infrastructure," arXiv:2602.23036, 2026.

APPENDIX A
EXPANDED GLOBAL CROSSOVER MATRIX

Rate	Mean input	Out 16	Out 32	Out 64	Out 128	Out 256	Out 512	Out 1024	Out 2048
4 rps	128	C4 (+29)	C4 (+64)	C4 (+118)	C4 (+221)	C4 (+444)	C4 (+895)	C4 (+1879)	P1/D3 (-2331)
4 rps	256	C4 (+31)	C4 (+64)	C4 (+114)	C4 (+211)	C4 (+413)	C4 (+835)	C4 (+1766)	P1/D3 (-2560)
4 rps	512	C4 (+35)	C4 (+65)	C4 (+110)	C4 (+185)	C4 (+353)	C4 (+714)	C4 (+1534)	P1/D3 (-3011)
4 rps	1024	C4 (+44)	C4 (+73)	C4 (+104)	C4 (+148)	C4 (+241)	C4 (+469)	C4 (+1071)	P1/D3 (-3911)
4 rps	1536	C4 (+48)	C4 (+79)	C4 (+106)	C4 (+111)	C4 (+130)	C4 (+218)	C4 (+594)	P1/D3 (-4797)
4 rps	2048	C4 (+52)	C4 (+75)	C4 (+101)	C4 (+69)	C4 (+31)	P1/D3 (-25)	C4 (+91)	P1/D3 (-5684)
4 rps	3072	C4 (+63)	C4 (+71)	C4 (+45)	P2/D2 (-11)	P2/D2 (-111)	P1/D3 (-440)	P1/D3 (-821)	P1/D3 (-7301)
4 rps	4096	C4 (+81)	C4 (+69)	P2/D2 (-18)	P2/D2 (-172)	P2/D2 (-435)	P2/D2 (-946)	P2/D2 (-1438)	P1/D3 (-7905)
4 rps	6144	C4 (+66)	C4 (+47)	P2/D2 (-64)	P2/D2 (-440)	P2/D2 (-1161)	P2/D2 (-2477)	P2/D2 (-4512)	P2/D2 (-11070)
4 rps	7168	C4 (+56)	P3/D1 (-8)	P3/D1 (-173)	P3/D1 (-473)	P2/D2 (-1412)	P2/D2 (-3168)	P2/D2 (-5938)	P2/D2 (-12561)
4 rps	10000	C4 (+41)	P3/D1 (-123)	P3/D1 (-511)	P3/D1 (-1379)	P3/D1 (-3069)	P3/D1 (-5588)	P3/D1 (-7981)	P2/D2 (-13171)

TABLE A.1

EXPANDED GLOBAL TOKEN-SHAPE CROSSOVER MATRIX FOR A FOUR-GPU SERVER AT 4 REQ/S, EXTENDED TO 10000 MEAN INPUT TOKENS AND 2048 MEAN OUTPUT TOKENS. CELLS SHOW THE GLOBAL WINNER AMONG COLOCATED DP (C4) AND PD SPLITS P1/D3, P2/D2, AND P3/D1. EACH WINNER IS THE CANDIDATE WITH THE LOWEST MEAN E2E LATENCY IN THAT TOKEN-SHAPE CELL. PARENTHESES REPORT BEST-PD-MINUS-C4 MEAN E2E LATENCY IN MILLISECONDS; NEGATIVE VALUES MEAN THE BEST PD SPLIT IS FASTER THAN C4. AT THIS HIGHER REQUEST RATE, THE 10000-TOKEN PROMPT ROW KEEPS P2/D2 AS THE BEST GLOBAL WINNER FOR THE 2048-OUTPUT CORNER, WHILE VERY LARGE PROMPTS MAKE P3/D1 WIN SEVERAL SHORTER-OUTPUT CELLS. ALL CELLS ARE EXTRAPOLATED.