PRESAGE: BOTTLENECK-AWARE TWO-STAGE SCHEDULING FOR DISAGGREGATED LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

PD-disaggregated LLM serving splits request scheduling into two decisions: inter-instance routing of prefill-completed requests, and intra-instance admission into the running batch. Both commit resources for a duration set by the request's remaining output length, which is unknown at decision time, so existing systems fall back on myopic heuristics over observable state, which produces multidimensional load imbalance across decode instances and preemption cascades within them.

We present Presage, a two-stage scheduler that couples both decisions through a shared distributional predictor. A sampling-aware predictor fuses intermediate LLM activations with sampling parameters into a five-class output-length distribution, improving cosine similarity by 16.5%–47.9% over the matched TRAIL++ and μ-Serve baselines; it runs concurrently with prefill at a median latency of 12.7 ms and, for 99.8% of requests, completes within the prefill duration, adding no end-to-end latency. Guided by this distribution, a dual-path router spends decision effort in proportion to load: below saturation it takes a sub-microsecond token-reservation fastpath, while near saturation, where routing errors are costly, it projects load onto compute, memory-bandwidth, and capacity pressure. Within each decode instance, a TPOT-driven admission controller tracks the measured per-token latency and adapts its admission window to it, halving the window when latency exceeds its target and growing it otherwise. On 8 A800 GPUs across three production traces, two model families, and two PD ratios, Presage achieves 1.02$\times$ to 2.17$\times$ the goodput of round-robin and NVIDIA Dynamo, with the largest gains where prefix-cache affinity creates hotspots.

1 INTRODUCTION

PD-disaggregated serving (Zhong et al., 2024; Qin et al., 2025; Patel et al., 2024) places compute-bound prefill and memory-bound decode on separate instances to eliminate interference. Disaggregation, however, splits scheduling into two decisions made at different pipeline stages: *inter-instance routing* assigns each prefill-completed request to a decode instance, and *intra-instance admission* decides when it joins the running batch. Both commit resources for a duration set by how many tokens the request will still generate, which is unknown when the decision must be made.

Existing systems hedge this uncertainty with myopic heuristics, and both stages suffer. Figure 1 makes the routing failure concrete: on one shared state, round-robin (a) follows the rotation and lands a long request on an already-busy instance, queue-length (b) mistakes low occupancy for free capacity, and prefix-cache (c) concentrates load where cache affinity is highest; all three overload an instance already committed to a long request. Routing on the predicted length (d) reserves the predicted margin and places both arriving requests where they fit. Admission (Figure 2) fails the same way: greedy admission (a) overfills and repeatedly preempts the overflow, conservative pre-allocation (b) reserves the maximum length and leaves memory idle, while ours (c) reserves the predicted length, the hatched margin, so four requests fit with nothing preempted, the window A itself adapted to measured per-token latency.

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

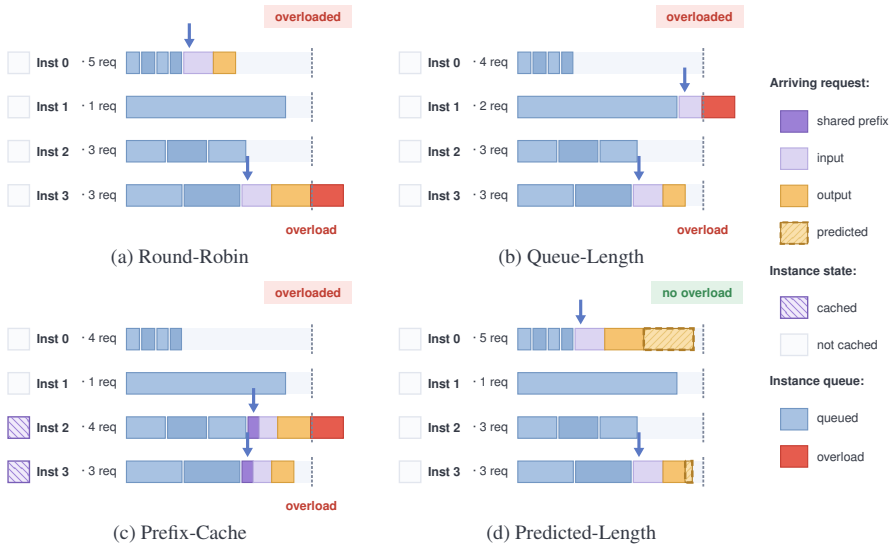


Figure 1: Inter-instance routing on one shared state: a row is a decode instance, a blue block a queued request (width = output still owed), the dashed line its limit, the red block the overflow. Two arriving requests (short, long) are routed by round-robin (a), queue length (b), prefix cache (c), and predicted length (d); only (d) reserves the hatched margin and stays within all four limits.

Both failures reduce to one latent variable, future output length and its uncertainty, suggesting we couple the two stages through one predictive signal. Three challenges arise: prediction must stay off the critical path (no post-prefill or extra forward pass); a point estimate is inadequate because output length is stochastic under sampling, and one mispredicted scalar suffices to misroute; and a predicted length must be projected onto the resource dimension that will actually bind, with accuracy mattering most near saturation.

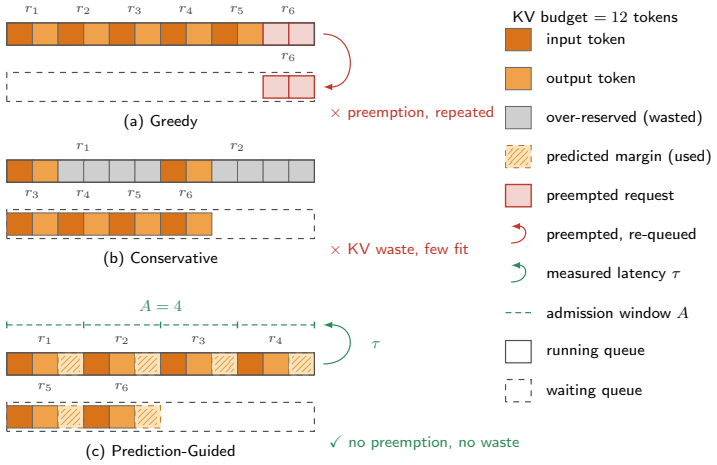


Figure 2: Intra-instance admission on one shared KV budget: greedy (a), conservative (b), and prediction-guided (c).

We present Presage, a two-stage scheduler with three components. A *sampling-aware predictor* fuses intermediate prefill activations with sampling-parameter embeddings into a five-class length distribution, running concurrently with prefill at a median 12.7 ms; against the activation-only baseline it lifts accuracy from 0.461 to 0.682 and cosine similarity from 0.678 to 0.852 on Llama, and from 0.458 to 0.547 and 0.605 to 0.705 on Qwen. *Dual-path routing* spends effort in proportion to slack: a sub-microsecond reservation fastpath below saturation, a projection onto compute, bandwidth, and capacity near it. A *geometric-window admission controller* adapts its window to measured per-token

latency, halving it when the target is exceeded and growing it by an eighth otherwise, and refines each remaining-length estimate at $O(1)$ cost.

Across three production traces, two model families, and two PD ratios, Presage attains 1.02×–2.17× the goodput of round-robin and NVIDIA Dynamo. One distributional signal thus coordinates routing and admission, a coordination neither prediction-agnostic nor unidimensional schedulers achieve.

2 Background and Related Work

LLM inference and PD disaggregation. Transformer LLMs (Vaswani et al., 2017) serve a request in two phases: prefill processes the prompt in parallel (time-to-first-token, TTFT) and decode generates autoregressively (time-per-output-token, TPOT). Prefill is compute-bound and decode memory-bound, so DistServe (Zhong et al., 2024) and Splitwise (Patel et al., 2024) disaggregate them; Sarathi-Serve (Agrawal et al., 2024) chunks long prefills and FlashAttention (Dao et al., 2022) shrinks attention. Disaggregation makes routing explicit, and existing routers all reason from observed state: round-robin (Holmes et al., 2024; Moritz et al., 2018; NVIDIA Corporation, 2025c), post-hoc imbalance repair (Sun et al., 2024; Wang et al., 2026), and prefix-cache affinity (Qin et al., 2025; Hu et al., 2025). None can tell which resource a new request will saturate.

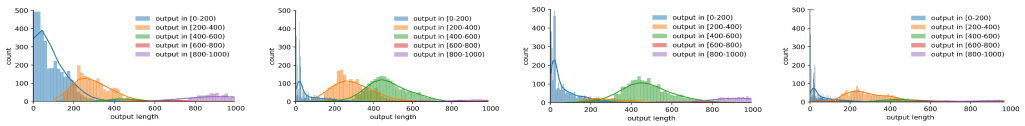
Intra-instance scheduling. vLLM (Kwon et al., 2023) builds on Orca's (Yu et al., 2022) continuous batching and admits greedily against current KV occupancy; DeepSpeed-MII (Holmes et al., 2024) pre-allocates for the maximum length. FastServe (Wu et al., 2023), μ-Serve (Qiu et al., 2024a), and TRAIL (Shahout et al., 2024) use predicted lengths to prioritize short requests, and Efficient Batch (Cheng et al., 2024) reorders within a batch. These govern the *order* in which queued requests run, not how many run concurrently, so the admitted volume that determines preemption cascades stays reactive to instantaneous memory.

Output-length prediction. Predictors differ in input and granularity: PastFuture (Gong et al., 2025) extrapolates from completed requests, S3 (Jin et al., 2023) and μ-Serve (Qiu et al., 2024a) fine-tune prompt encoders such as BERT (Devlin et al., 2019), TRAIL (Shahout et al., 2024) reads the serving model's intermediate activations, and others use proxy models (Hu et al., 2025; Qiu et al., 2024b), random forests (Cheng et al., 2024), or instruction-tuned LLMs (Zheng et al., 2023). All infer length from content or history alone, ignoring the sampling parameters that shift end-of-sequence probability, and emit a point estimate that discards the uncertainty a scheduler needs for tail risk. Presage closes both gaps with a sampling-aware distributional predictor that drives routing and admission from one shared signal.

3 Preliminary Study

Existing predictors share two gaps: they ignore sampling parameters, and they emit point estimates. This section quantifies the first gap and validates the predictor that closes it.

Why sampling parameters matter. Sampling parameters determine how logits become tokens and thus the end-of-sequence probability at each step: temperature rescales logit sharpness, top-p (nucleus sampling (Holtzman et al., 2020)) and top-k truncate the candidate set, and the repetition penalty demotes generated tokens. Figure 3 shows temperature alone shifting the output-length distribution markedly across an otherwise identical request set; a predictor blind to these parameters cannot separate such cases.



(a) Temperature = 0.2 (b) Temperature = 0.3 (c) Temperature = 0.5 (d) Temperature = 0.9

Figure 3: Output-length distributions (five buckets) of a fixed request set on Llama-3-8B-Instruct as sampling temperature varies from 0.2 to 0.9 (a–d).

Setup. We record 10000 real generations of Llama-3-8B-Instruct and Qwen-2.5-7B-Instruct on LMSYS-Chat-1M (Zheng et al., 2024) (prompt, middle-layer activations, attention scores, ground-truth length; 6:2:2 train/validation/test). Table 1 reports five-bucket classification (accuracy, F1,

cosine) and regression (MAE) via separate heads over one encoding; the requests replayed in the evaluation are disjoint.

Table 1: Prediction accuracy on Llama-3-8B-Instruct and Qwen-2.5-7B-Instruct (classification by accuracy/F1/cosine, regression by MAE); SE = sampling-parameter embeddings.

Method	Input	Architecture	Filter	Layer	Accuracy	F1	Cosine	MAE
Llama-3-8B-Instruct								
μ-Serve	prompt	BERT-based	head-tail	-	0.4615	0.4170	0.5760	-
TRAIL	activation	BERT-based	head-tail	16	0.4580	0.4332	0.6766	-
TRAIL++	activation	BERT-based	attention score	16	0.4610	0.4320	0.6780	-
Presage	activation	BERT-based + SE	attention score	16	**0.6820**	**0.6686**	**0.8520**	-
STAR	activation	MLP	last token	32	-	-	-	3178
Presage	activation	BERT-based + SE	attention score	16	-	-	-	**2713**
Qwen-2.5-7B-Instruct								
μ-Serve	prompt	BERT-based	head-tail	-	0.4335	0.3913	0.5653	-
TRAIL	activation	BERT-based	head-tail	14	0.4400	0.4226	0.5999	-
TRAIL++	activation	BERT-based	attention score	14	0.4575	0.4358	0.6054	-
Presage	activation	BERT-based + SE	attention score	14	**0.5470**	**0.5176**	**0.7053**	-
STAR	activation	MLP	last token	28	-	-	-	6566
Presage	activation	BERT-based + SE	attention score	14	-	-	-	**6036**

Component analysis. Table 1 isolates each component. Replacing prompt text with activations (μ-Serve $\rightarrow$ TRAIL) leaves top-1 accuracy essentially unchanged while raising cosine similarity markedly on both models, confirming that activations expose length signal absent from the prompt surface; attention-based token filtering (TRAIL $\rightarrow$ TRAIL++) moves it only marginally, which we report without claiming significance. Sampling embeddings (TRAIL++ $\rightarrow$ Presage) account for the dominant gain, lifting accuracy to 0.682 on Llama and 0.547 on Qwen, and regression improves in step, with MAE well below STAR's (Wang et al., 2026) on both models.

Why the distribution matters. Cosine similarity measures alignment of the predicted probability mass with the true bucket rather than top-1 correctness. Presage improves it by 16.5%–47.9% over the two baselines on both models, so the emitted distribution is usable as such: the router consumes it to weigh expected load against tail risk, which a point estimate cannot supply.

4 Motivation

Output length is predictable as a distribution, but both stages need that distribution rather than a scalar: routing must anticipate which resource a request will saturate (Observation 1), and admission must bound future growth the running batch can absorb (Observation 2). Experiments use A800-80GB GPUs, one per instance, under a 1000 ms TTFT and 50 ms TPOT SLO, with a tighter 30 ms SLO also examined.

4.1 Observation 1: Routing Must Anticipate Future Bottlenecks

Routing poses an inherently predictive question: which resource will bind once this request starts generating? Scalar heuristics collapse distinct dimensions: queue length and batch size are blind to KV pressure, and raw occupancy conflates capacity with bandwidth contention. Decode pressure is instead three-dimensional (concurrency, attention traffic, and the footprint of weights, activations, and KV cache), so we characterize an instance by three dimensionless indicators, each relating one decode step's demand to what the TPOT SLO affords:

$$L_C = \frac{F_{\text{decode}}}{F_{\text{TPOT}}}, \qquad L_M = \frac{M_{\text{decode}}}{M_{\text{TPOT}}}, \qquad L_K = \frac{C_{\text{used}}}{C_{\text{total}}}, \tag{1}$$

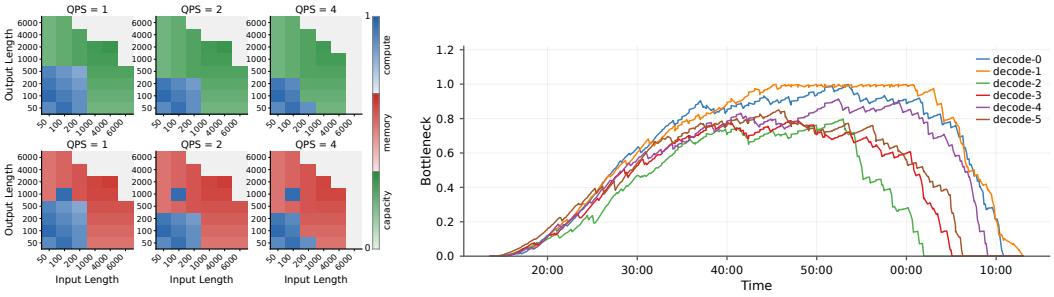
where F_{decode} and M_{decode} are the per-step FLOPs and memory-traffic bytes (KV cache, activations, and weights), F_{TPOT} and M_{TPOT} the per-step FLOPs and memory traffic the TPOT SLO affords, and C_{used}/C_{total} the occupied and total device memory; these capture compute, bandwidth, and capacity pressure. The dominant bottleneck is

$$L_{\text{bottle}} = \max(L_C, L_M, L_K), \tag{2}$$

which Presage instantiates with online metrics.

The dominant bottleneck is not fixed. In our 1P3D runs, bandwidth dominates as requests arrive, capacity overtakes it as KV cache accumulates, and bandwidth returns once completions release it. Sweeping input and output length under 1P1D (Figure 4a) shows the binding resource set jointly by input length, output length, request rate, and TPOT SLO: short inputs and outputs at high rates are compute-bound, long ones push the bottleneck toward bandwidth or capacity. Routing thus cannot assume a persistent bottleneck.

Myopic routing creates multidimensional imbalance. Figure 4b shows the six decode instances' bottleneck loads diverging, with round-robin keeping some instances above saturation while others idle, so routers blind to future bottlenecks systematically induce imbalance.



(a) Dominant bottleneck regimes

(b) Cross-instance load imbalance

Figure 4: The binding resource is neither fixed nor balanced. (a) Dominant bottleneck regime over input/output length, request rate, and TPOT SLO (Llama-3-8B-Instruct, 1P1D, 1000 prompts per cell; top row TPOT $\leq$ 50 ms, bottom $\leq$ 30 ms; gray cells denote out-of-memory configurations). (b) Bottleneck load of each decode instance under round-robin (Qwen-2.5-7B-Instruct, 1683 ShareGPT prompts, 2P6D, 50 ms TPOT SLO).

Length information recovers the loss, but only as a distribution. An oracle routing each request to the instance with the fewest future tokens outperforms round-robin (Figure 5a); substituting point predictions under the same rule leaves a residual gap (Figure 5b), because point predictors are brittle on long or domain-specific prompts and output length is stochastic under sampling. Routing therefore requires a future bottleneck model driven by a length *distribution*.

Dataset		Lmsys-chat-1M			
Req/s		6	8	10	12
Llama-3-8B-Instruct	Output	2.09x	2.50x	2.10x	1.79x
	Total	2.08x	2.48x	2.08x	1.77x

(a) Oracle gain

Dataset		Lmsys-chat-1M			
Req/s		6	8	10	12
Llama-3-8B-Instruct	Output	2.48x	2.48x	2.20x	2.20x
	Total	2.48x	2.48x	2.20x	2.20x

(b) Prediction gap

Figure 5: Value of length information for routing under 1P3D (TTFT 1000 ms, TPOT 50 ms). (a) Oracle-to-round-robin goodput ratio, Llama-3-8B-Instruct with LMSYS-Chat-1M prompts at 6–12 req/s; other models and datasets in Appendix C. (b) Oracle-to-point-prediction ratio on the same model and prompts, the point predictor using the mean of 50 sampled generations.

4.2 Observation 2: Admission Must Control Future Interference

Once a request lands, the scheduler decides when it joins the running batch; every running request enlarges its KV cache, so admitting more improves utilization until interference triggers preemption. The decision depends on future growth, not current occupancy, since two requests with identical footprints today may differ sharply in what they will still generate. Figure 6 examines this on Llama-3-8B-Instruct under 1P1D across four workload compositions and two memory budgets (vLLM GPU utilization 0.3 and 0.8); solid lines report goodput, dashed lines throughput.

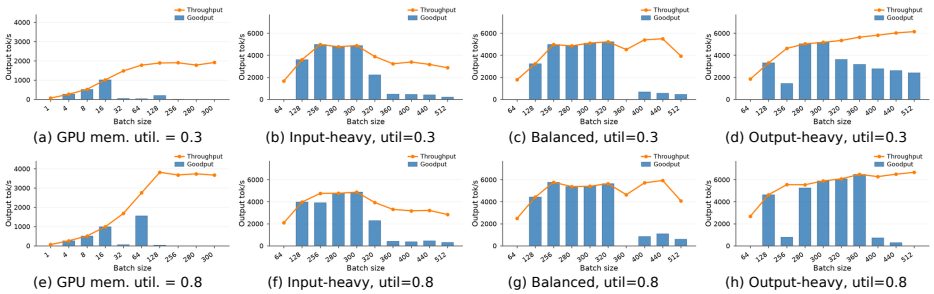


Figure 6: Goodput (solid) and throughput (dashed) across workload compositions. Llama-3-8B-Instruct, 1P1D; (b–d),(f–h) hold 160000 tokens under different input–output splits; top row GPU utilization 0.3, bottom 0.8; columns: fixed, input-heavy 120k/40k, balanced 80k/80k, output-heavy 40k/120k.

Admission has an optimal concurrency region. With fixed 512-input/1024-output (columns a,e), goodput traces an inverted U under both budgets, while throughput stays high well after goodput collapses, so throughput is a misleading admission target. The peak moves with the budget, so the admission target must track available budget rather than remain fixed.

Token count alone is insufficient. Columns b–d and f–h hold exactly 160000 tokens and vary only the input–output split; were admission difficulty a function of total tokens, the curves would coincide, but they do not. Output-heavy batches keep requests resident and accumulate KV cache faster, shifting the peak; identical token counts thus yield different outcomes depending on composition and budget.

Admission therefore cannot rely on current token counts: it must account for how tokens are distributed and how much each request may still generate. With Observation 1, this yields three requirements: a predictor on the serving path emitting a distribution rather than a scalar, a router converting that distribution into projected three-dimensional pressure and assigning on post-admission load, and an instance scheduler consuming the same signal to trade utilization against preemption risk. Both stages hinge on one latent variable, so both should be driven by one shared signal.

5 Design

5.1 Overview

Presage realizes these requirements in the pipeline shown in Figure 7: the predictor runs concurrently with prefill and emits a five-class length distribution; the proxy routes on a three-dimensional load snapshot (a reservation fastpath below saturation, a projected-length comparison near it); and a TPOT-driven scheduler adapts its admission window inside each instance, closing the loop.

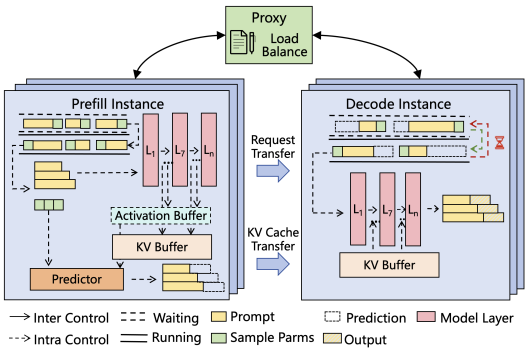


Figure 7: Presage architecture. The predictor runs concurrently with prefill; the proxy routes on three-dimensional load, and each decode instance admits under TPOT-driven control.

5.2 SAMPLING-AWARE PREDICTOR

The predictor maps per-request signals to a five-class distribution in three steps (Figure 8). First, attention scores select the 512 most informative tokens from a middle layer during prefill (shorter inputs are left unfiltered), and the retained activations are projected to the hidden dimension of a bert-base-uncased encoder (110M parameters) and framed with [CLS] and [SEP]:

$$\mathbf{h}^{(0)} = [\mathbf{c}_{\text{cls}};\ \mathbf{W}_{\text{proj}}\mathbf{a};\ \mathbf{c}_{\text{sep}}], \quad \mathbf{W}_{\text{proj}} \in \mathbb{R}^{d_{\text{bert}} \times d_{\text{llm}}}. \tag{3}$$

Second, the encoder's [CLS] representation $\mathbf{h}_{\text{cls}}$ is fused additively with learned embeddings of temperature, top-p, top-k, and repetition penalty:

$$\mathbf{z} = \mathbf{h}_{\text{cls}} + \sum_{s \in \{t,p,k,r\}} \text{Embed}_s(v_s). \tag{4}$$

Third, an MLP head produces the bucket distribution

$$\mathbf{q} = \text{softmax}(\mathbf{W}_3 \text{ReLU}(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{z}))), \quad \mathbf{q} \in \mathbb{R}^5, \tag{5}$$

trained end-to-end with cross-entropy. The five buckets partition the model's maximum supported context length into equal fifths, and b_k is the midpoint of bucket k; the scheduler consumes $\mathbf{q}$ together with these centers, never a single argmax, so both stages reason about tail risk.

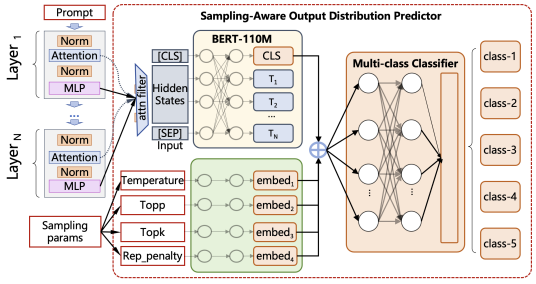


Figure 8: Predictor architecture. Attention-selected activations are encoded, fused with sampling-parameter embeddings, and classified into five length buckets.

5.3 DUAL-PATH INTER-INSTANCE ROUTING

Instantiating the load model. We instantiate the indicators L_C, L_M, L_K as online ratios of per-step demand to what the TPOT SLO affords. Compute load follows from projection and attention FLOPs, bandwidth load from weight, KV-cache, and activation reads, and capacity load from KV-cache utilization; Appendix A derives the per-step counts, and the bottleneck is their maximum (Equation 2).

Projecting the future. Routing needs post-admission pressure, so for instance i we accumulate

$$\text{FutureTokens}(i) = \sum_{r \in \mathcal{R}_{\text{run}}(i)} p_r + \sum_{r \in \mathcal{R}_{\text{wait}}(i)} (\ell_r^{\text{prompt}} + p_r), \tag{6}$$

where p_r is predicted remaining length for running requests and predicted total for waiting ones.

Choosing a path. Presage spends effort in proportion to slack (Algorithm 1). Each instance exposes headroom R_i in its tightest dimension; Path A fires only when some instance is genuinely idle ($R_i \geq \Delta$, TPOT below θ_{route}, no waiting request, KV below θ_{kv}) and routes to $\arg\max_i R_i$, debiting Δ. Otherwise Path B awaits the predictor, shortlists the better half of candidates, and selects the lowest current bottleneck (disjoint from inference).

5.4 GWF INTRA-INSTANCE SCHEDULER

Geometric window feedback. Presage runs *Geometric Window Feedback* (GWF, Algorithm 2): it tracks τ, an EMA of per-token latency (over requests with $\geq s_{\min}$ samples), and every Δt halves the admission window A when τ exceeds the target θ, or grows it by an eighth otherwise.

Ranking and admission. Requests are ranked for eviction by hopelessness (TTFT or TPOT budget exceeded), then by eviction value $\text{eV}(r)$; a request that cannot allocate KV is preempted and returned

Algorithm 1 Dual-Path Inter-Instance Routing

Require: $\mathcal{D}$: decode instances; $\mathcal{L}$: load snapshots; Δ: reservation quantum; θ_{route}: measured-TPOT idle gate; θ_{kv}: KV-occupancy idle gate; l_p: prompt length; b_p, b_c: projected and current bottleneck
Ensure: i^*: selected instance

for each $i \in \mathcal{D}$ **do**
  $R_i \leftarrow \min(\text{comp_rem}_i, \text{mem_rem}_i, \text{kv_rem}_i) - \text{reserved}_i$
end for
if $\exists i : R_i \geq \Delta \ \wedge\ \text{TPOT}_i < \theta_{\text{route}} \ \wedge\ \text{wait}_i = 0 \ \wedge\ \text{kv}_i < \theta_{\text{kv}}$ **then**
  $i^* \leftarrow \arg\max_i R_i$; $R_{i^*} \leftarrow R_{i^*} - \Delta$; $\text{reserved}_{i^*} \leftarrow \text{reserved}_{i^*} + \Delta$
  AssignDecode(i^*);
  return i^* {Path A}
end if
AwaitPrefillAndPredictor(); $l_o \leftarrow$ ExpectedLength()
$l \leftarrow l_p + \lceil l_o \rceil$; $\mathcal{C} \leftarrow \emptyset$
for each healthy $i \in \mathcal{D}$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\text{PredictiveLoad}(\mathcal{L}[i], l),\ \mathcal{L}[i].\text{bottleneck},\ i)\}$
end for
keep top $\lceil |\mathcal{C}|/2 \rceil$ by b_p, then $i^* \leftarrow \arg\min_{b_c}(\mathcal{C})$ {Path B}
AssignDecode(i^*);
return i^*

to the queue. Waiting requests are then admitted under the window, which demotes hopeless requests and promotes those deferred beyond $D_{\max}$, so admission follows measured TPOT rather than a static memory budget.

Refining the prediction online. A prefill-time prediction grows stale, so Presage refines it from the same distribution: after g generated tokens the remaining length is

$$p_r(g) = \sum_{k=1}^{5} q_k \max(0, b_k - g). \tag{7}$$

This residual is monotone non-increasing and $O(1)$ to update, so one prediction serves both stages.

Algorithm 2 GWF Intra-Instance Scheduler

Require: $\mathcal{R}_{\text{run}}, \mathcal{R}_{\text{wait}}$: running/waiting requests; $A \in [A_{\min}, A_{\max}]$: admission window; θ: TPOT target; $D_{\max}$: deferral bound; $s_{\min}$: sample floor; Δt: update period; $\text{eV}(r)$: eviction value
Ensure: schedOutput

$\tau \leftarrow \text{EMA}\{\text{tpot}(r) : r \in \mathcal{R}_{\text{run}},\ \text{samples}(r) \geq s_{\min}\}$
every Δt: **if** $\tau > \theta \ \wedge\ |\mathcal{R}_{\text{run}}| \geq A$ **then** $A \leftarrow \max(A_{\min}, A/2)$
  elif $\tau \leq \theta$ **then** $A \leftarrow \min(A_{\max}, A + A/8)$
sort $\mathcal{R}_{\text{run}}$ by $(\text{hopeless}(r),\ \text{eV}(r))$ descending; *preempted* $\leftarrow$ false
for each $r \in \mathcal{R}_{\text{run}}$ **do**
  if $\text{allocKv}(r) =$ None **then** preempt r; $\mathcal{R}_{\text{wait}}.\text{add}(r)$; *preempted* $\leftarrow$ true
  else schedule r
end for
if $\neg$*preempted* **then**
  for each $r \in \mathcal{R}_{\text{wait}}$ **do**
    if $|\mathcal{R}_{\text{run}}| \geq A$ **then break**
    if $\text{hopeless}(r) \ \wedge\ |\mathcal{R}_{\text{run}}| \geq A/2$ **then** Demote(r)
    elif $\text{deferral}(r) \geq D_{\max}$ **then** Promote(r); admit r
    elif $\text{allocKv}(r) \neq$ None **then** admit r
  end for
end if
return schedOutput

6 Evaluation

Our evaluation asks whether coupling routing and admission through a shared predictor improves goodput, how much of that gain is prediction rather than load awareness, and whether the predictor's cost stays off the serving path.

6.1 Experimental Setup

We follow Appendix B: 8 A800-80GB GPUs under **2P6D/4P4D** vs. **Round-Robin** and **Dynamo** (NVIDIA Corporation, 2025a); 3 traces $\times$ 4 workloads; goodput in output tok/s under a 1000 ms TTFT and 30 ms TPOT SLO, median of three runs.

6.2 End-to-End Goodput



Figure 9: Output token goodput across 24 trace–workload–PD combinations under a 30 ms TPOT SLO; left 2P6D, right 4P4D.

Figure 9 reports all 24 combinations: Presage leads on every Mooncake workload (Dynamo's prefix-cache affinity creates imbalance outweighing its reuse), peaks at 2.17$\times$ round-robin on Qwen, and converges under BurstGPT's bursts (1.02$\times$–2.17$\times$ overall).

6.3 Routing Ablation and Predictor Overhead

We compare four routers on Llama-3-8B-Instruct/LMSYS-Chat-1M under 2P6D, holding the scheduler fixed: **Round-Robin**, **P2D-Capacity**, **P2D-3DLoad** (3D snapshot), and **Presage-3DLoad** (adds the predictor). Capacity routing beats rotation by 15.3% (1862 to 2147 tok/s), 3D state adds 0.4% (2155 tok/s), and the predictor adds 32.8% to 2861 tok/s, a cumulative 53.7% over Round-Robin; its own median latency of 12.7 ms stays within the prefill tail for 1679 of 1683 requests, so it adds nothing to the critical path (Appendix D).

6.4 Discussion and Limitations

Prediction-guided routing beats prefix-cache affinity whenever load is heterogeneous, resolving cache reuse against balance rather than always favoring reuse. Our evaluation is nonetheless bounded: it covers two 7–8B models, three traces, and one SLO, ablates routing without isolating the admission controller, and reports aggregate goodput rather than latency distributions. The predictor is also trained offline per model family, so cross-model transfer and online adaptation remain open, all the more so because Section 4 showed that oracle lengths still outperform any deployed predictor.

7 Conclusion

PD-disaggregated serving splits scheduling into routing and admission, both committing resources against a request's unknown remaining output length, so both should be driven by one signal. Presage couples them through one distributional predictor: a sampling-aware length model, a dual-path router projecting it onto compute, bandwidth, and capacity, and a TPOT-driven admission controller. It attains 1.02$\times$–2.17$\times$ the goodput of round-robin and Dynamo, and closing the gap to oracle routing is the next step.

AI Use Statement

During manuscript preparation, the authors used generative artificial intelligence (AI) tools for language editing, manuscript organization, figure and table refinement, LaTeX debugging, and assistance with the plotting and analysis scripts used to generate the figures. AI tools were also used to assist with parts of the system code development, including code completion and debugging. The research

questions, method and scheduler designs, benchmark methodology, and all experimental measurements are the authors' own, and AI-assisted suggestions were checked against the source data and implementation. The authors take responsibility for the final text, figures, claims, and experimental results.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.

Ke Cheng, Wen Hu, Zhi Wang, Peng Du, Jianguo Li, and Sheng Zhang. Enabling efficient batch serving for LMaaS via generation length prediction. In *Proceedings of the 2024 IEEE International Conference on Web Services (ICWS)*, pp. 853–864, 2024.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90% ChatGPT quality. https://lmsys.org/blog/2023-03-30-vicuna/, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 16344–16359, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171–4186, 2019.

Ruihao Gong, Shihao Bai, Siyu Wu, Yunqian Fan, Zaijun Wang, Xiuhong Li, Hailong Yang, and Xianglong Liu. Past-future scheduler for LLM serving under SLA guarantees. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 25)*, pp. 798–813, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference. *arXiv preprint arXiv:2401.08671*, 2024.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

CunChen Hu, HeYang Huang, LiangLiang Xu, XuSheng Chen, Chenxi Wang, Jiang Xu, Shuang Chen, Hao Feng, Sa Wang, Yungang Bao, et al. ShuffleInfer: Disaggregate LLM inference for mixed downstream workloads. *ACM Transactions on Architecture and Code Optimization*, 22(2), 2025. doi: 10.1145/3732941.

Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. s^3: Increasing GPU utilization during generative inference for higher throughput. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 18015–18027, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP 23)*, pp. 611–626, 2023.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 561–577, 2018.

NVIDIA Corporation. NVIDIA Dynamo: A datacenter-scale distributed inference serving framework. https://github.com/ai-dynamo/dynamo, 2025a.

NVIDIA Corporation. NVIDIA Collective Communications Library (NCCL). https://github.com/NVIDIA/nccl, 2025b.

NVIDIA Corporation. NVIDIA Triton Inference Server. https://github.com/triton-inference-server/server, 2025c.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st ACM/IEEE International Symposium on Computer Architecture (ISCA 24)*, pp. 118–132, 2024.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—A KVCache-centric architecture for serving LLM chatbot. In *Proceedings of the 23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, 2025.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Power-aware deep learning model serving with μ-Serve. In *Proceedings of the 2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 75–93, 2024a.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Efficient interactive LLM serving with proxy model-based sequence length prediction. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 24)*, 2024b.

Rana Shahout, Eran Malach, Chunwei Liu, Weifan Jiang, Minlan Yu, and Michael Mitzenmacher. Don't stop me now: Embedding based scheduling for LLMs. *arXiv preprint arXiv:2410.01035*, 2024.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191, 2024.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. Towards efficient and reliable LLM serving: A real-world workload study. *arXiv preprint arXiv:2401.17644*, 2024.

Zhibin Wang, Zetao Hong, Xue Li, Zibo Wang, Shipeng Li, Qingkai Meng, Qing Wang, Chengying Huan, Rong Gu, Sheng Zhong, et al. STAR: Decode-phase rescheduling for LLM inference. In *Proceedings of the 35th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '26)*, 2026. doi: 10.1145/3806645.3807813.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-based generative models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

ZeroMQ Community. ZeroMQ: Distributed messaging. https://zeromq.org, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. LMSYS-Chat-1M: A large-scale real-world LLM conversation dataset. *arXiv preprint arXiv:2309.11998*, 2024.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: An LLM-empowered LLM inference pipeline. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.

A Derivation of the Decode-Side 3D Load Model

This appendix instantiates the three indicators L_C (compute), L_M (bandwidth), and L_K (capacity) introduced in Section 4. L, h, I, V, and K denote the number of decoder layers, hidden dimension, FFN intermediate dimension, vocabulary size, and KV projection dimension (head dimension times the number of KV heads); β is the bytes per KV element; m_r is the tokens cached for request r; and $\mathcal{R}_{\text{run}}$ and $\mathcal{R}_{\text{wait}}$ are the running and waiting sets. $P_{\max}$ is the peak GPU throughput in FLOPs/s, $B_{\max}$ the peak device bandwidth in bytes/s, and TPOT_{ms} the TPOT SLO in milliseconds.

A.1 Compute Pressure (L_C)

Each decode step drives one token per request through all linear projections and the output head, while attention reads the request's full cache. The linear FLOPs per step are

$$\text{FLOPs}_{\text{linear}} = 2L\big(h(2h+2K)+3hI\big)+2Vh, \tag{8}$$

and the attention FLOPs for request r are

$$\text{FLOPs}_{\text{attn}}(r) = 4Lh\,m_r. \tag{9}$$

Normalizing by the peak throughput times the TPOT SLO gives

$$L_C = \sum_{r\in\mathcal{R}_{\text{run}}} \frac{\text{FLOPs}_{\text{linear}}+\text{FLOPs}_{\text{attn}}(r)}{P_{\max}\,\text{TPOT}_{\text{ms}}\,10^{-3}}. \tag{10}$$

A.2 Memory Bandwidth Pressure (L_M)

Each decode step reads the model weights once (shared across the running batch) and, per request, its KV cache and activations. The shared weight reads amount to

$$W_{\text{weights}} = \big(L\,[h(2h+2K)+3hI]+Vh\big)\,\beta, \tag{11}$$

and the per-request KV-cache and activation bytes are

$$\text{KV}_{\text{bytes}}(r) = 2LKm_r\beta, \qquad \text{Act}_{\text{bytes}} = L(2h+2I)\beta, \tag{12}$$

so the bandwidth load is

$$L_M = \frac{W_{\text{weights}}+\sum_{r\in\mathcal{R}_{\text{run}}}\big(\text{KV}_{\text{bytes}}(r)+\text{Act}_{\text{bytes}}\big)}{B_{\max}\,\text{TPOT}_{\text{ms}}\,10^{-3}}. \tag{13}$$

A.3 Capacity and Bottleneck (L_K)

The capacity load is the fraction of device memory occupied; since the KV cache dominates decode-phase memory, this reduces to the utilization of the KV-cache blocks,

$$L_K = \frac{\mathrm{KV}_{\mathrm{used}}}{\mathrm{KV}_{\mathrm{total}}}, \tag{14}$$

and the dominant bottleneck is the maximum of the three, per Equation 2.

B Detailed Experimental Setup

Traces. We replay segments of three production traces with contrasting arrival patterns: **Qwen** (n = 4080, T = 600 s) sustains a dense, high request rate; **Mooncake** (n = 1790) (Qin et al., 2025) represents moderate load; and **BurstGPT** (n = 673) (Wang et al., 2024) exhibits pronounced bursts. Pairing Llama-3-8B-Instruct (Grattafiori et al., 2024) and Qwen-2.5-7B-Instruct (Team, 2024) with prompts from LMSYS-Chat-1M (Zheng et al., 2024) and ShareGPT (Chiang et al., 2023) yields 24 trace-workload-PD combinations.

Predictor training. The predictor is trained offline per model family on 10000 real generations recorded from LMSYS-Chat-1M, split 6:2:2 into training, validation, and test sets, for 16 epochs at batch size 32 with early stopping on the validation set. The requests replayed in Section 6 are disjoint from these 10000 and never used for training or selection.

Scheduler configuration. Routing reserves Δ tokens per fastpath decision, set to half the model's maximum sequence length, and takes the fastpath only for an instance that is idle by the gates $\theta_{\mathrm{route}} = 120$ ms of measured TPOT and $\theta_{\mathrm{kv}} = 0.6$ of KV occupancy. The intra-instance window A is bounded by $A_{\min} = 32$ and $A_{\max} = 256$, updated every $\Delta t = 1$ s against a target $\theta = 120$ ms on a TPOT EMA that requires $s_{\min} = 4$ samples per request, and a request demoted as hopeless is force-promoted after $D_{\max} = 2000$ ms. We choose θ as a reachable operating point for the window feedback rather than the 30 ms TPOT SLO, which is enforced separately by the eviction and demotion logic of Section 5 against the 30 ms budget.

Hardware. Experiments run on 8 NVIDIA A800-80GB SXM4 GPUs connected by NVLink and NVSwitch, with inter-GPU communication over NCCL (NVIDIA Corporation, 2025b), one GPU per instance, with the proxy on a dedicated CPU node communicating over ZeroMQ (ZeroMQ Community, 2025). Baselines (Round-Robin, Dynamo (NVIDIA Corporation, 2025a)) share the same hardware, checkpoints, and vLLM backends.

Software. We use vLLM v0.11.0 as the serving backend and NVIDIA Dynamo v0.7.1 for the Dynamo baseline, on Python 3.10 with PyTorch 2.8.0, CUDA 12.8, NCCL 2.27.3, and transformers 4.56.1.

C Full Oracle-Routing Gain

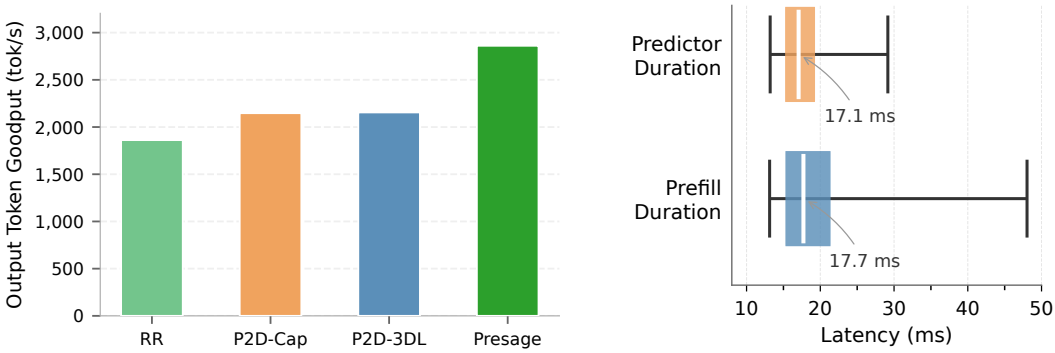
Figure 10 shows the complete oracle-to-round-robin goodput ratio behind Figure 5(a), covering both models and both datasets; the gain is consistent across Qwen-2.5-7B-Instruct and ShareGPT, so the main text shows only the Llama-3-8B-Instruct LMSYS-Chat-1M result.

Dataset		LMSys-Chat				ShareGPT			
	Req/s	6	8	10	12	6	8	10	12
Llama	Output	**2.09x**	**2.50x**	**2.10x**	**1.79x**	**1.04x**	**1.05x**	**1.07x**	**1.06x**
	Total	**2.08x**	**2.48x**	**2.08x**	**1.77x**	**1.04x**	**1.05x**	**1.07x**	**1.06x**
Qwen	Output	**1.19x**	**1.20x**	**1.27x**	**1.23x**	**1.73x**	**1.39x**	**1.15x**	**1.57x**
	Total	**1.19x**	**1.19x**	**1.26x**	**1.21x**	**1.70x**	**1.38x**	**1.15x**	**1.55x**

Figure 10: Complete oracle-to-round-robin goodput ratio (output and total token goodput) on Llama-3-8B-Instruct and Qwen-2.5-7B-Instruct, 1000 prompts each from LMSYS-Chat-1M and ShareGPT, across request rates of 6–12 req/s.

D Routing Ablation and Predictor Overhead

Figure 11 gives the routing ablation and predictor-overhead results behind Section 6.3.



(a) Routing ablation

(b) Latency hiding

Figure 11: Routing ablation and predictor overhead (Llama-3-8B-Instruct, LMSYS-Chat-1M, 2P6D): (a) goodput of Round-Robin, P2D-Capacity, P2D-3DLoad, Presage-3DLoad; (b) predictor latency vs. prefill tail (medians 12.7/14.1 ms, P95 14.6/20.1 ms; $n = 1683$).