SIBYL: Speculative Inference Balancing for XPYD Scheduling in Disaggregated LLM Serving

Anonymous Authors[1]

Abstract

Disaggregated serving is widely adopted for large language model inference, where requests are assigned to decode instances at prefill start to enable overlapping of KV-cache transmission with computation. The assignment typically aims to maximize performance based on the current loads of decode instances. However, when a request finishes prefilling, the actual load on its assigned decode instance usually differs from that at the time of decision-making. This *load discrepancy* makes the assignment suboptimal and often leads to elevated tail latency that violates service-level objectives. The discrepancy stems from two sources: (1) backlog requests may finish decoding before the new request arrives at the assigned instance; and (2) requests currently undergoing prefilling may transition to decoding at any moment. To address this issue, we propose SIBYL, a speculative scheduler that projects the loads at the time when the new request starts decoding, and assigns it to the instance with the least projected load. Experiments on real clusters and large-scale simulations demonstrate that SIBYL reduces decoding tail latency by up to 78% compared to state-of-the-art approaches.

1. Introduction

The rapid adoption of large language models (LLMs) (Yang et al., 2025; OpenAI et al., 2024b; Team et al., 2025b; Grattafiori et al., 2024; MiniMax et al., 2025; Team et al., 2025a; Comanici et al., 2025; Zhao et al., 2025; Minaee et al., 2025) has driven the need for high-throughput, low-latency inference systems. LLM inference proceeds in two stages: (1) *prefill*, which processes the full prompt to initialize the key-value cache (KV Cache), and (2) *decode*,

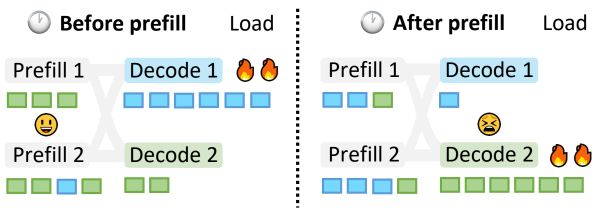


Figure 1. **The load discrepancy in disaggregated serving.** *Each colored block denotes a request, and the color indicates the decode instance it is bound to before prefill. A assignment made at prefill start (Left) becomes suboptimal at decode start (Right) as the system state evolves during prefill, causing load imbalance.*

which autoregressively generates tokens while reading the KV cache. To improve resource utilization and scalability, many modern serving frameworks (Zhong et al., 2024; Patel et al., 2025; Qin et al., 2025; DeepSeek-AI et al., 2025; Kwon et al., 2023; SGLang, 2025; NVIDIA, 2025) adopt *disaggregated serving*, which separates prefill and decode into two instance[1] pools with X prefill instances and Y decode instances (XPYD): prefill instances process prompts and produce KV caches, while decode instances consume these caches to generate tokens. The separation of prefill and decode eliminates the mutual interference in mixed workloads, where bursty prefills and concurrent decodes competitively stall each other (Agrawal et al., 2024). This disaggregated design is critical for production-scale serving to meet service level objectives (SLOs) for both time-to-first-token (TTFT) and time-per-output-token (TPOT).

Despite the architectural benefits, disaggregated clusters frequently suffer from severe tail latency, causing SLO violations (Wang et al., 2025; Zhang et al., 2025). This tail-latency issue stems from load imbalance: some instances become overloaded while others remain underutilized. The root cause is a fundamental *load discrepancy* introduced by early assignment, which aims to overlap KV cache transmission with computation (Qin et al., 2025; Patel et al., 2025). This means the scheduler must choose a target decode instance at prefill start, while decoding only begins at the handoff moment (when prefill completes), risking placement on an instance that becomes overloaded during the prefill interval. As illustrated in Figure 1, a scheduler

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]Throughout this paper, an *instance* hosts a full model replica.

relying on the instantaneous snapshot at prefill start (Left) routes the request to the seemingly idle Decode 2. However, by the time decoding begins (Right), the cluster state has inverted. Decode 2 becomes congested by concurrent arrivals, while Decode 1 (previously busy) frees its resources, making the assignment significantly suboptimal.

Existing schedulers fail to mitigate this discrepancy because they rely on instantaneous metrics that become outdated during the prefill interval (Kwon et al., 2023; SGLang, 2025; Li et al., 2025; NVIDIA, 2025; Heisler et al., 2025). Consequently, solving this problem requires shifting from reactive placement to predictive look-ahead. However, modeling the exact cluster state at the future handoff moment is non-trivial due to two distinct sources of uncertainty: (1) *Residual Uncertainty*: The remaining lifetime of currently active tasks, which is highly stochastic, especially with the variance in generation lengths; and (2) *Shadow Interference*: Requests currently undergoing prefill, which constitute a latent workload that will contend for decode resources at the future handoff moment. Navigating these uncertainties to enable precise early assignment remains an unresolved challenge.

To mitigate the load discrepancy, we introduce SIBYL, a speculative scheduler for disaggregated serving. SIBYL projects per-instance load at the handoff moment and early-assigns each request to the instance with the minimum projected load. To compute this projection, SIBYL (1) estimates the residual lifetimes of active decode tasks via survival analysis, and (2) accounts for shadow interference from requests still in prefill, which may enter decoding at any moment.

We make the following contributions:

- We identify a *load discrepancy* between the scheduling (prefill start) and execution (decode start) as the root cause of load imbalance in disaggregated serving.
- We propose SIBYL, a speculative scheduler with a survival-analysis-based workload model. It combines physical pressure with survival probability to predict load at the future handoff moment, enabling look-ahead balancing while preserving early-assignment benefits.
- We demonstrate that SIBYL reduces tail latency by up to 78% over state-of-the-art baselines in real-cluster experiments and improves scalability in large-scale simulations.

2. Preliminaries

2.1. The Heterogeneity of Disaggregated Serving

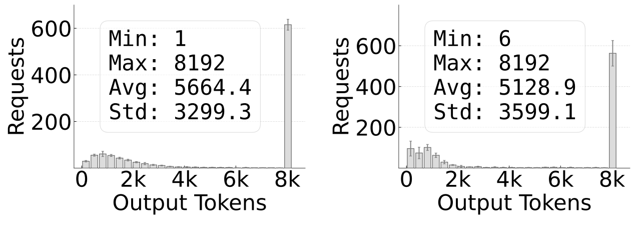
LLM inference has two phases: *prefill*, which processes the full input prompt in parallel, and *decode*, which autoregressively generates tokens step-by-step. These phases have fundamentally different hardware affinities: prefill is compute-bound and bursty, while decode is memory-bandwidth-bound and latency-sensitive (Jain et al., 2025b;a). Modern serving systems (Patel et al., 2025; Zhong et al., 2024; Qin et al., 2025; NVIDIA, 2025; Jin et al., 2024; Singh et al., 2025; Ma et al., 2025) adopt disaggregated serving, where prefill and decode workloads are conducted by dedicated pools of instances, backed by heterogeneous hardware tailored to each phase's characteristics. This architecture also enables independent scaling of prefill and decode clusters, allowing each phase to be scaled up or down according to its distinct SLO requirements. This flexibility has driven its widespread adoption in production environments (Qin et al., 2025; Xiang et al., 2025; DeepSeek-AI et al., 2025).

2.2. The Load Discrepancy in Disaggregated Serving

Disaggregated serving introduces a load discrepancy between scheduling and execution. To make KV-cache transmission in parallel with computation, requests must be assigned to a decode instance before prefill starts. The load of decode instances, however, can change drastically after the prefill period, due to the influence of "uncertain" evolving loads. We identify two sources of this uncertainty:

(1) Residual Uncertainty from Active Tasks. At the moment of scheduling, instances host ongoing decode tasks with uncertain remaining lifetimes. This is further exacerbated by the emergence of reasoning-intensive models (*e.g.,* Chain-of-Thought (Wei et al., 2023; OpenAI et al., 2024a)), where generation length is dictated by the complexity of the logic chain rather than simple conversational patterns. We validate this using Qwen3-32B (Yang et al., 2025), as shown in Figure 2. The workloads exhibit extreme variance driven by reasoning steps: ShareGPT (ShareGPT Teams, 2023) displays a massive deviation ($\sigma \approx 3317$) with an average output length of 5657 tokens, frequently saturating the 8192 token limit. The Zeta dataset (Zed Industries, 2024) exhibits a similarly large standard deviation ($\sigma \approx 3492$). As a result, residual decode load is inherently difficult to predict, often leading to severe error of future load.

(2) Interference from Shadow Tasks. Equally critical is the *shadow workload*, which refers to requests assigned to a decode instance but still currently undergoing prefill. Although not yet decoding, these tasks impose two distinct



(a) ShareGPT *(b)* Zeta

Figure 2. **Output lengths on different datasets using Qwen3-32B (Yang et al., 2025).** *The workloads are evaluated with reasoning enabled and a maximum output limit of 8192 tokens.*

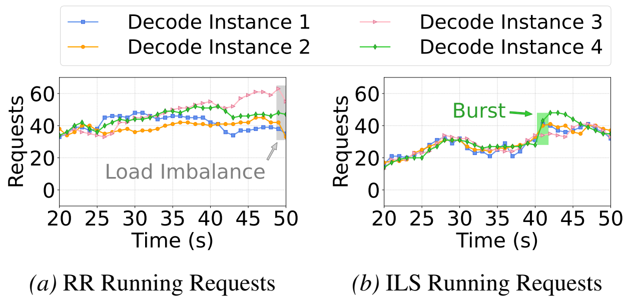


Figure 3. **Limitations of baseline scheduling policies under a ShareGPT workload.** *(a) RR fails to account for request heterogeneity, leading to significant load imbalance. (b) ILS ignores the prefill-decode time gap, blindly routing multiple requests to the same seemingly idle instance, which causes transient load spikes.*

risks to the given new request upon decoding: tasks transitioning to the decode phase *prior* to the new request (impose immediate contention) and those joining the decode phase during the mid-execution (introduce delayed interference). Failing to account for these risks blinds the scheduler to impending pressures, causing concurrent requests to converge on the same seemingly idle instance.

2.3. Limitations of Existing Solutions

Most existing schedulers are myopic, relying on static rules or instantaneous load snapshots, and thus become unreliable under stochastic, fast-evolving workloads (Figure 3). Migration can partially correct imbalance, but it is reactive and introduces non-trivial transfer overhead. We next summarize representative strategies and their limitations.

Round-Robin (RR). Being load-agnostic, RR (Kwon et al., 2023) fails to accommodate the heavy-tailed heterogeneity of LLM generation. As shown in Figure 3(a), the stochastic arrival of long-decode tasks can lead to disproportionate backlog accumulation on specific instances. Since RR continues to blindly route requests to these instances, it exacerbates load imbalance and leads to severe tail latency.

Instantaneous Load Scheduling (ILS). Strategies such as least outstanding requests (Li et al., 2025; NVIDIA, 2025) route new tasks to the instance with the minimum instantaneous load. This approach is myopic because it ignores the temporal gap between the assignment decision (prefill start) and actual execution (prefill end). Under bursty traffic, multiple requests are routed to the same seemingly idle instance. As they finish prefill and start decoding, they trigger the sharp congestion spikes visible in Figure 3(b).

Rescheduling and Migration. While migration mechanisms (Sun et al., 2024; Wang et al., 2025) can mitigate imbalance, they are inherently reactive and costly. They incur high bandwidth penalties from inter-node KV transfer and mandate invasive modifications to engine internals (*e.g.,* vLLM or SGLang), rendering them impractical for standard black-box deployments.

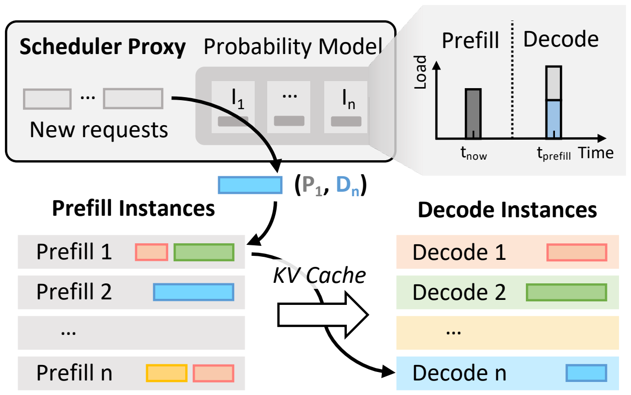


Figure 4. **System overview of SIBYL.** *The probability-driven global scheduler proxy can predict future decode loads. It assigns new requests to prefill (P_1) and decode (D_n) instances based on projected availability at the estimated handoff time.*

Overall, existing designs are either myopic or migration-heavy, motivating SIBYL as a proactive, lightweight method to mitigate load discrepancy in disaggregated serving.

3. Design Overview

3.1. System Architecture

Figure 4 illustrates the architecture of SIBYL. Implemented as a global scheduler proxy, the system advances beyond instantaneous load balancing by maintaining a *Probabilistic Workload Model* that tracks the state of every active request to forecast the occupancy evolution of decode instances. The core workflow enforces future-load-aware early assignment. Upon request arrival, the proxy calculates the prefill duration, which can be considered deterministic given the prompt length (Xiang et al., 2025; Zhong et al., 2024; Lou et al., 2025). Leveraging this duration, SIBYL projects the cluster state at the future handoff moment (t_{prefill}) and binds the request to the decode instance (D_n) that minimizes future imbalance. This predictive assignment enables the overlap of KV-cache transmission with computation while ensuring load balance at the actual start of decoding.

3.2. Design Philosophy: "Bridging" the Temporal Gap

To mitigate the load discrepancy, SIBYL optimizes for the projected cluster state at the future handoff moment τ rather than the current snapshot t_{now}. As shown in Figure 5, the scheduler aggregates potential load (introduced by three types of tasks) to identify the instance with the minimal projected load at the handoff moment τ:

- **Type 1: Active Tasks.** Requests currently being decoded. Their residual load at τ is stochastic, derived from the survival probability during prefill.
- **Type 2: Pre-Handoff Shadow.** Concurrent prefill

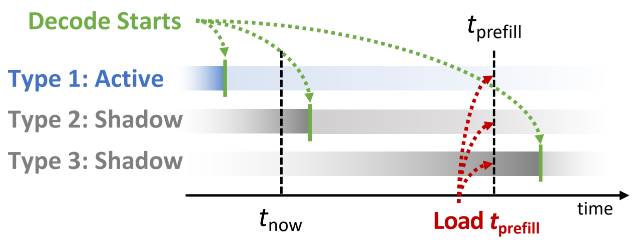


Figure 5. **Speculative load projection.** *The scheduler estimates instance occupancy at τ (the future handoff moment, also referred to as t_{prefill}), when the new tasks arrive at t_{now}, by aggregating the residual active load and the shadow load from pending tasks.*

tasks projected to start decoding *before* τ. These form hidden contention that is invisible at t_{now} but will be occupying resources at the handoff.

- **Type 3: Post-Handoff Shadow.** Concurrent prefill tasks projected to start decoding *after* τ. Although arriving later, they pose a potential interference risk that may constrain near-term availability.

By aggregating the projected impact of these workloads, SIBYL constructs a comprehensive view of the *future* load.

4. Probabilistic Workload Modeling

We establish a probabilistic framework to formulate the scheduling decision as a load minimization problem at the handoff moment. For reference, we summarize the key notations in Appendix B.

4.1. Objective Function

Let $\mathcal{D}$ denote the set of decode instances. For a candidate instance $j \in \mathcal{D}$, the total projected load $\mathcal{L}_j(\tau)$ at the future handoff time τ is defined as the sum of the residual load from currently running tasks and the projected load from pending tasks (*i.e.,* requests currently undergoing prefill that are already bound to instance j):

$$\mathcal{L}_j(\tau) = \sum_{k \in \mathcal{Q}_j^{\text{run}}} \mathcal{L}_{\text{I}}(k,\tau) + \sum_{k \in \mathcal{Q}_j^{\text{pend}}} \left(\mathbb{I}_{\tau_k \le \tau} \mathcal{L}_{\text{II}}(k,\tau) + \mathbb{I}_{\tau_k > \tau} \mathcal{L}_{\text{III}}(k,\tau) \right) \tag{1}$$

where $\mathcal{Q}_j^{\text{run}}$ and $\mathcal{Q}_j^{\text{pend}}$ denote the sets of active and pending tasks on instance j, respectively. The terms $\mathcal{L}_{\text{I}}$, $\mathcal{L}_{\text{II}}$, and $\mathcal{L}_{\text{III}}$ correspond to the projected loads of active, pre-handoff shadow, and post-handoff shadow tasks, as detailed in §3.2. $\mathbb{I}_{\text{condition}}$ denotes the indicator function, which equals 1 if the condition holds and 0 otherwise. Based on this metric, the scheduler identifies the optimal instance $j^* = \arg\min_{j \in \mathcal{D}} \mathcal{L}_j(\tau)$, thereby directing the request to the instance with the minimum projected load at the handover instant. The following sections detail the modeling of the individual load terms $\mathcal{L}_{\text{I, II, III}}$.

4.2. Modeling Primitives

To compute $\mathcal{L}_{\text{I, II}}$ in Eq. (1), we introduce a mechanistic model built upon two primitives: *Physical Pressure* ($\mathcal{M}$) and *Survival Probability* ($\mathcal{P}$).

Primitive 1: Physical Pressure ($\mathcal{M}$). We define physical pressure $\mathcal{M}_k(l)$ as the request's total token occupancy to quantify its hardware cost:

$$\mathcal{M}_k(\tau) = l_k^{\text{in}} + l_k(\tau) \tag{2}$$

where l_k^{in} denotes the prompt length and $l_k(\tau)$ represents the generated length at time τ. Since LLM decoding is memory-bandwidth bound (Zhong et al., 2024; Patel et al., 2025), the execution overhead is dominated by loading the KV cache, which scales linearly with the total sequence length. Thus, $\mathcal{M}$ effectively captures the hardware cost.

Primitive 2: Conditional Survival Probability ($\mathcal{P}$). This component quantifies the *likelihood* of a request remaining active at a specific future moment. Predicting exact generation lengths is inherently unstable (Wu et al., 2024; Wang et al., 2025). Instead, we model task completion uncertainty using the survival function, which directly captures the probability of a request remaining active beyond a given time. Let L_k denote the generation length of request k. We define the survival function as $\mathcal{S}(x) = \mathbb{P}(L_k > x)$ for a length threshold x, and assume that $\{L_k\}$ are i.i.d. samples drawn from a common distribution. For a request k with current generated length $l_k(t_{\text{now}})$, the probability $\mathcal{P}_k(\tau)$ that it survives up to the projected generated $l_k(\tau)$ is:

$$\begin{aligned}\mathcal{P}_k(\tau) &= \mathbb{P}(L_k > l_k(\tau) \mid L_k > l_k(t_{\text{now}})) \\ &= \frac{\mathbb{P}(L_k > l_k(\tau))}{\mathbb{P}(L_k > l_k(t_{\text{now}}))} = \frac{\mathcal{S}(l_k(\tau))}{\mathcal{S}(l_k(t_{\text{now}}))}\end{aligned} \tag{3}$$

To compute Eq. (3) practically, we approximate the theoretical function $\mathcal{S}$ using an online estimator $\hat{\mathcal{S}}$. Instead of tracking every integer length, we discretize the output length space into coarse-grained intervals (buckets) with a fixed step size Δ. The estimator entries $\hat{\mathcal{S}}(l)$ are maintained only for discrete boundaries $l \in \{\Delta, 2\Delta, \dots\}$. Upon each request completion with final length L_{final}, we update the estimated probabilities for all relevant buckets ($l \le L_{\text{final}}$) using an exponential moving average (EMA):

$$\hat{\mathcal{S}}(l) \leftarrow \alpha \cdot \hat{\mathcal{S}}(l) + (1-\alpha) \cdot \mathbb{I}_{L_{\text{final}} \ge l} \tag{4}$$

where $\alpha \in (0,1)$ is the smoothing factor. This online update prioritizes recent observations, allowing the estimator to refine $\hat{\mathcal{S}}$ to reflect the current output length distribution.

4.3. Speculative Load Composition

Using these primitives, we define the projected load for tasks potentially active at τ (*i.e.,* Type 1 and Type 2) as the expected sequence length at τ, as established in Eq. (5).

$$\mathcal{L}(k,\tau) = \mathcal{M}_k(\tau) \cdot \mathcal{P}_k(\tau), \qquad k \in \text{Type 1/2}. \tag{5}$$

We provide the formal derivation justifying this definition in Appendix C. In contrast, Type 3 tasks do not begin decoding until after τ, contributing zero instantaneous load at the handoff. Consequently, instead of applying Eq. (5), we model $\mathcal{L}_{\text{III}}(k,\tau)$ as a lookahead interference penalty to capture near-future contention.

Type 1: Active Tasks ($t_{\text{start}} \leq t_{\text{now}}$). For tasks already in decoding, we leverage their observed execution statistics for projection. We project the token count using the task's specific decoding rate v_k, and compute the conditional survival probability directly from our estimator $\hat{\mathcal{S}}$:

$$\mathcal{L}_{\text{I}}(k,\tau) = \underbrace{(l_k^{\text{in}} + l_k(\tau))}_{\mathcal{M}} \cdot \underbrace{\frac{\hat{\mathcal{S}}(l_k(\tau))}{\hat{\mathcal{S}}(l_k(t_{\text{now}}))}}_{\mathcal{P}} \tag{6}$$

where $l_k(\tau) = l_k(t_{\text{now}}) + v_k \cdot (\tau - t_{\text{now}})$ is obtained by linearly projecting the generated length using the task's observed effective decoding rate v_k over the short interval $[t_{\text{now}}, \tau]$. The probability term $\mathcal{P}$ captures the likelihood that task k remains active from its current generation length $l_k(t_{\text{now}})$ to the projected generation length $l_k(\tau)$.

Type 2: Pre-Handoff Shadow Tasks ($\tau_k \leq \tau$). Let τ_k denote the projected start time of task k. We model pending tasks satisfying $\tau_k \leq \tau$ as pre-handoff shadow tasks. Since these tasks become active before the new request arrives, they will occupy resources at moment τ. Unlike Type 1 tasks where per-task decoding rate is observable, pending tasks lack runtime history. We rely on the system-wide average speed $\bar{v}_{\text{sys}}$ to estimate the potential resource consumption, and define the projected generated length as $l_k(\tau) = (\tau - \tau_k) \cdot \bar{v}_{\text{sys}}$. For Type 2 tasks, the probability term is computed from the start state at τ_k, where $l_k(\tau_k) = 0$. Thus, the conditional survival probability up to $l_k(\tau)$ reduces to $\hat{\mathcal{S}}(l_k(\tau))/\hat{\mathcal{S}}(0)$, which simplifies to $\hat{\mathcal{S}}(l_k(\tau))$ since $\hat{\mathcal{S}}(0) = 1$. Putting everything together, we obtain the expected load for Type 2 tasks:

$$\mathcal{L}_{\text{II}}(k,\tau) = \underbrace{(l_k^{\text{in}} + l_k(\tau))}_{\mathcal{M}} \cdot \underbrace{\hat{\mathcal{S}}(l_k(\tau))}_{\mathcal{P}} \tag{7}$$

Type 3: Post-Handoff Shadow Tasks ($\tau_k > \tau$). For tasks projected to start after the handoff, we use an *interference-decay* model to capture their near-future contention with the post-handoff execution window. Although these tasks are inactive at the handoff moment, they will start decoding shortly after and may overlap with the decode of the current request, thereby introducing additional interference. We model this interference by treating the prompt length l_k^{in} as the base interference. We then linearly discount this based on the temporal distance $(\tau_k - \tau)$, using the system-average decode rate $\bar{v}_{\text{sys}}$. Intuitively, tasks starting soon after τ have limited temporal slack and are more likely to overlap with ongoing requests, whereas later arrivals experience weaker contention. We therefore clamp the discounted value at zero to ignore sufficiently distant arrivals:

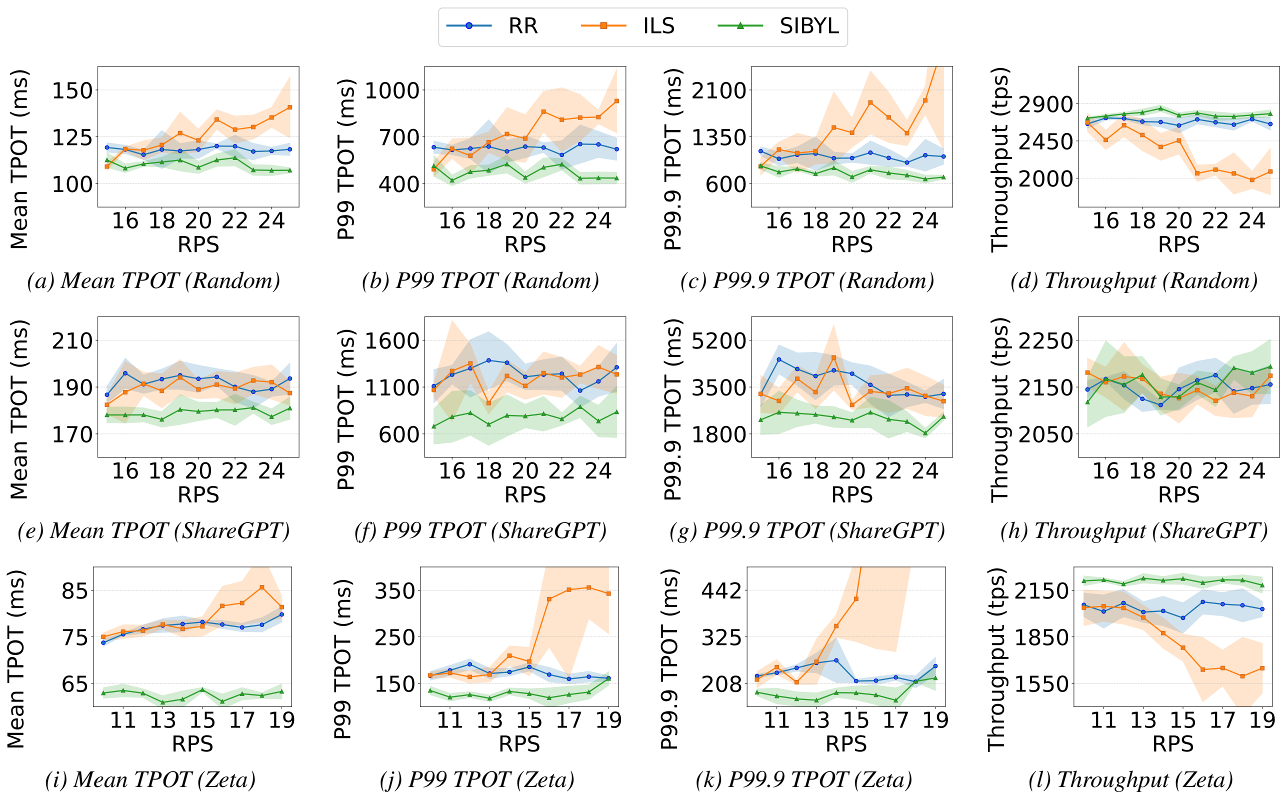
$$\mathcal{L}_{\text{III}}(k,\tau) = \text{ReLU}\big(l_k^{\text{in}} - \bar{v}_{\text{sys}}\,(\tau_k - \tau)\big). \tag{8}$$

Algorithm 1 SIBYL

```
Input : Request $r$, Time $t_{\text{now}}$, Instances $\mathcal{D}$, Estimator $\hat{\mathcal{S}}$
1  Function ScheduleRequest($r, t$):
2      $\delta_{\text{pre}} \leftarrow \textsc{EstPrefill}(r.length)$
3      $\tau \leftarrow t_{\text{now}} + \delta_{\text{pre}}$
4      for each instance $j \in \mathcal{D}$ do
5          $\mathcal{L}_j \leftarrow 0$
6          for $k \in \mathcal{Q}_j^{run}$ do
7              $l_k(\tau) \leftarrow l_k(t_{\text{now}}) + v_k \cdot (\tau - t)$
8              $\mathcal{M} \leftarrow l_k^{\text{in}} + l_k(\tau)$
9              $\mathcal{P} \leftarrow \hat{\mathcal{S}}(l_k(\tau))/\hat{\mathcal{S}}(l_k(t_{\text{now}}))$
10         for $k \in \mathcal{Q}_j^{pend}$ do
11             $\tau_k \leftarrow k.arrival + \textsc{EstPrefill}(k.length)$
12             $\Delta_{\text{gap}} \leftarrow (\tau - \tau_k) \cdot \bar{v}_{sys}$
13             if $\Delta_{gap} > 0$ then
14                 $\mathcal{M} \leftarrow l_k^{\text{in}} + \Delta_{\text{gap}}$
15                 $\mathcal{P} \leftarrow \hat{\mathcal{S}}(\Delta_{\text{gap}})$
16             else
17                 $\mathcal{M} \leftarrow \max(0, l_k^{\text{in}} - \Delta_{\text{gap}})$
18                 $\mathcal{P} \leftarrow 1$
19             $\mathcal{L}_j \leftarrow \mathcal{L}_j + (\mathcal{M} \times \mathcal{P})$
20     return $\arg\min_j \mathcal{L}_j$
```

4.4. Scheduling Algorithm

Algorithm 1 outlines the scheduling procedure and runs in $O(|\mathcal{D}| + N_{\text{total}})$ time. Here, $|\mathcal{D}|$ is the number of candidate decode instances, and N_{total} is the number of requests traversed for load aggregation (active decodes and pending shadows). The procedure begins by establishing the target handoff time τ using the new request's estimated prefill latency (Line 2). For each candidate instance, the algorithm then aggregates the projected load from two distinct sources. First, it projects the state of currently active tasks (Lines 5–9), weighting their estimated physical pressure by the corresponding conditional survival probability; Second, it evaluates pending shadow tasks (Lines 11–20) by estimating their start times and separately handling: Type 2 expected loads ($\tau_k \leq \tau$) and Type 3 future-load penalties ($\tau_k > \tau$). Finally, the algorithm selects the instance that minimizes the resulting cumulative congestion (Line 20).



(a) *Mean TPOT (Random)* (b) *P99 TPOT (Random)* (c) *P99.9 TPOT (Random)* (d) *Throughput (Random)*
(e) *Mean TPOT (ShareGPT)* (f) *P99 TPOT (ShareGPT)* (g) *P99.9 TPOT (ShareGPT)* (h) *Throughput (ShareGPT)*
(i) *Mean TPOT (Zeta)* (j) *P99 TPOT (Zeta)* (k) *P99.9 TPOT (Zeta)* (l) *Throughput (Zeta)*

Figure 6. End-to-end performance comparison on three dataset. TPOT refers to time-per-output-token.

5. Evaluation

We first summarize the experimental settings in this section. Full details are provided in Appendix F.

Testbed. We conduct our evaluation on a server node with eight NVIDIA H20 GPUs. To accommodate the varying prefill-decode ratios of workloads, we deploy two cluster configurations: a 2P4D setup (2 prefill and 4 decode instances) for the random dataset, and a 4P4D setup for the ShareGPT and Zeta datasets. In all scenarios, each GPU hosts a replica of a single model. For the interconnect, we leverage vLLM's native `P2pNcclConnector` (NVIDIA Corporation, 2026) to enable efficient layer-wise KV cache transmission between prefill and decode instances.

Models. We evaluate SIBYL using Qwen3-32B (Yang et al., 2025), a widely adopted open-weights model in production environments. This model represents a typical single-GPU workload that fits comfortably within the H20's on-device memory (141 GB), eliminating the communication overhead of tensor parallelism (Shoeybi et al., 2020).

Datasets. We evaluate SIBYL across three distinct datasets to cover a range of inference scenarios. First, we utilize `vLLM bench` to generate a synthetic random workload with uniformly distributed input and output lengths for micro-benchmarking system performance. Second, we employ ShareGPT (ShareGPT Teams, 2023), a de facto standard dataset comprising real-world user conversations, which exhibits the heavy-tailed length distribution typical of chat services. Finally, we include Zeta (Zed Industries, 2024), a dataset of predictive code-editing tasks from Zed Industries. It is used to assess performance on specialized programming assistant workloads.

Baselines. We compare SIBYL against two standard policies: (1) *Round-Robin (RR)* (Kwon et al., 2023), a static, load-agnostic approach that dispatches requests cyclically; and (2) *Instantaneous Load Scheduling (ILS)* (Li et al., 2025; NVIDIA, 2025), a greedy policy that routes each request to the currently least-loaded instance.

5.1. Overall System Performance

Figure 6 summarizes the performance evaluation of SIBYL, ILS, and RR across three representative workloads. First, on the Random workload, SIBYL increases output token throughput by 1.21× (vs. ILS) and 1.03× (vs. RR), while reducing Mean TPOT by 12.0% (vs. ILS) and 6.7% (vs. RR). Notably, SIBYL effectively suppresses tail latency, lowering P99 and P99.9 TPOT by 32.7% and 43.4% (vs. ILS), as well as by 24.5% and 25.2% (vs. RR). Second, on the real-world ShareGPT dataset, while throughput remains comparable across methods due to system saturation

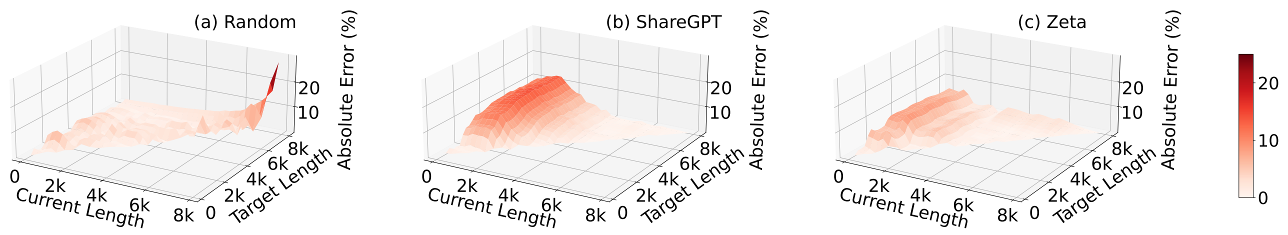


Figure 7. Calibration error of survival probability estimation.

(∼1.0×), SIBYL stabilizes generation performance by mitigating hotspots: it reduces Mean TPOT by ∼6% and cuts tail latency drastically, lowering P99 TPOT by 34.3% (vs. ILS) and 36.3% (vs. RR), and P99.9 TPOT by 28.6% (vs. ILS) and 34.8% (vs. RR). Finally, on the Zeta dataset, SIBYL achieves its most pronounced gains, boosting throughput by 1.22× (vs. ILS) and 1.09× (vs. RR) and reducing Mean TPOT by 20.7% (vs. ILS) and 18.9% (vs. RR). More importantly, SIBYL achieves strict SLO compliance with massive reductions in tail latency, where P99 and P99.9 TPOT drop by 41.9% and 50.6% (vs. ILS), as well as 24.2% and 20.8% (vs. RR). These results collectively show that SIBYL's predictive scheduling not only maximizes cluster capacity in bursty scenarios but also ensures prompt responsiveness in production-grade workloads. Additional results under different setups are detailed in Appendix G.

5.2. Estimation Error of Survival Probability

To validate the estimation error of SIBYL's profiling mechanism, we conduct the following analysis. We performed a static evaluation using the checkpoints trained on the Random, ShareGPT, and Zeta datasets. Specifically, we queried the estimator on a dense grid of current lengths and target lengths across the entire trace, calculating the mean absolute error between the predicted survival probability $\mathcal{P}$ and ground-truth conditional probabilities derived from the cumulative distribution functions of the test datasets. Figure 7 visualizes the error surface across the prediction space. Overall, SIBYL demonstrates robust calibration quality across diverse workload distributions. Even under the high output variance of ShareGPT, the estimator maintains an absolute error below 14%, with the vast majority of the configuration space exhibiting $< 5\%$ deviation. Similarly, for the coding-intensive Zeta dataset, the error remains under 9%. The Random workload also shows negligible estimation error consistent with a uniform distribution, except for minor boundary effects at extreme context lengths.

5.3. Sensitivity Analysis

To validate the effectiveness of SIBYL's design choices, we conduct sensitivity analysis by selectively disabling

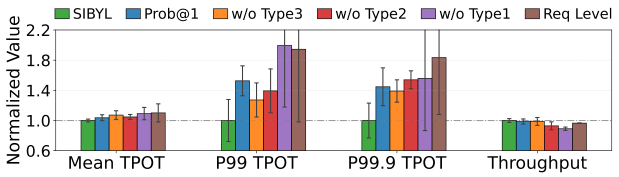


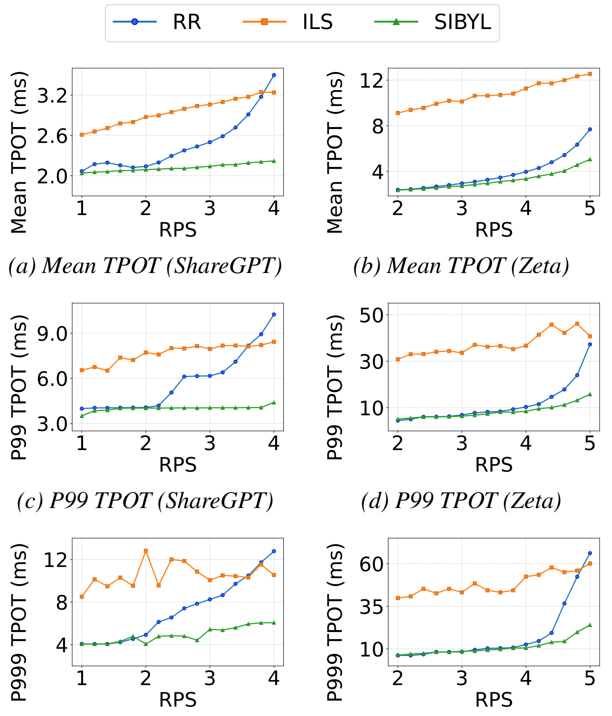
Figure 8. Sensitivity analysis of SIBYL components.

specific strategies or altering the load metric. Detailed methodology and variant definitions are provided in Appendix J. Figure 8 summarizes the results normalized to SIBYL (lower is better for TPOT). First, removing probabilistic weighting (PROB@1) increases P99/P99.9 TPOT by 1.53×/1.45×, showing the importance of modeling survival uncertainty. Second, removing Type 1 yields the largest P99 degradation (1.99×), followed by Type 2. This suggests that ongoing and soon-to-start decode requests dominate worst-case contention. Removing Type 3 leads to a smaller but consistent degradation, suggesting that post-handoff lookahead mainly stabilizes near-future interference. Third, replacing our token-based metric with simple request counts (REQLEVEL) results in severe tail latency degradation (1.83× P99.9 TPOT), demonstrating that counting requests is insufficient for heterogeneous LLM workloads.

5.4. Scalability Performance

To assess SIBYL's scalability in large-scale environments, we implemented a trace-driven discrete-event simulator modeling a cluster with 64 decode instances. The detailed methodology, along with extended evaluations on 128 and 256 instances, are provided in Appendix I.

Performance. Figure 9 shows the performance distribution. SIBYL consistently outperforms baseline strategies across all latency percentiles in large-scale settings. On the heavy-tailed ShareGPT workload, SIBYL improves Mean, P99, and P99.9 TPOT by 12.5%, 24.5%, and 24.8% compared to RR. The advantage over ILS is even more significant, yielding reductions of 28.0% in Mean, 47.7% in P99, and 53.0% in P99.9. The performance gap also shows on the Zeta dataset. Compare to RR, SIBYL reduces Mean, P99, and P99.9 TPOT by 12.7%, 14.4%, and 15.4%, respectively. Against ILS, SIBYL cuts Mean TPOT by 70.4% and



(a) Mean TPOT (ShareGPT) *(b) Mean TPOT (Zeta)* *(c) P99 TPOT (ShareGPT)* *(d) P99 TPOT (Zeta)* *(e) P99.9 TPOT (ShareGPT)* *(f) P99.9 TPOT (Zeta)*

Figure 9. Simulation on a cluster with 64 decode instances.

suppresses tail latency by 78.0% (P99) and 77.5% (P99.9). These results illustrate the scalability bottlenecks of baselines. RR's blind scheduling and ILS's "invisible load" issue lead to significant performance degradation at scale. In contrast, SIBYL's look-ahead scheduling effectively mitigates these inefficiencies.

Table 1. Optimal assignment ratio comparison.

Dataset	**RR**	**ILS**	**SIBYL**
ShareGPT	79.5%	71.6%	**94.2%** (↑1.19×)
Zeta	52.2%	22.6%	**64.5%** (↑1.24×)

Optimal Assignment Ratio. We further analyze the placement accuracy, defined as the probability that the pre-assigned instance remains the least-loaded node at the actual handoff moment. Table 1 reveals that ILS frequently underperforms RR (*e.g.,* 22.6% vs. 52.2% on Zeta), showing that stale snapshots cause concurrent requests to collide on the same seemingly idle instance. In contrast, SIBYL accurately predicts future states, achieving 94.2% accuracy on ShareGPT and more than doubling ILS's precision on Zeta. The lower accuracy on Zeta is due to its longer code inputs, which extend the prefill phase (widen the prediction window) and increase prediction uncertainty. This improved assignment precision directly explains the tail latency reductions observed in Figure 9.

6. Discussion

Learning-based Length Prediction. Recent works explore learning-based approaches that predict generation length to assist request placement (Wang et al., 2025; Jain et al., 2025b; Jin et al., 2023; Fu et al., 2024). While effective in controlled settings, these designs introduce additional prediction overhead on the scheduling path and require invoking a model for every request. Moreover, learned length prediction faces a robustness–overhead trade-off: higher accuracy typically costs more, while lightweight predictors are less reliable across workloads.

Robustness to Prediction Errors. SIBYL relies on lightweight profiling primitives, including coarse-grained decoding rates (v_k) and survival probabilities ($\mathcal{P}_k$). Importantly, these estimates are not required to be exact: the scheduler only depends on their relative ordering across instances, rather than precise numerical values. As a result, SIBYL can tolerate moderate estimation inaccuracies, as long as the projected load preserves the relative pressure across candidates. This design avoids the need for tightly calibrated predictors on the critical scheduling path.

Limitations and Future Work. (1) The effectiveness of SIBYL is workload-dependent. When decoding is short, prefill dominates latency, or decode resources are lightly loaded, the load discrepancy between scheduling and execution is limited, leaving less headroom for projection-based early assignment (Du et al., 2025). (2) Moreover, SIBYL adopts a per-request greedy strategy to enable fast and scalable scheduling decisions. While global optimization across concurrent arrivals may achieve better assignments under bursty workloads, it would incur higher coordination and computation overhead. (3) Finally, SIBYL focuses on mitigating load imbalance at the instance level. It does not explicitly address intra-instance imbalance, such as GPU-level skew introduced by expert parallelism (DeepSeek-AI, 2024; Doucet et al., 2025; Zhao et al., 2026). Incorporating finer-grained, device-level signals into load projection is a promising direction for future work.

7. Conclusion

We identify a fundamental *load discrepancy* in disaggregated LLM serving, where scheduling decisions made at prefill start often become stale by the time decoding begins. To address this issue, we present SIBYL, a speculative scheduler that projects per-instance load at the handoff moment using lightweight survival-based primitives. This projection guides load-aware early assignment and helps align early assignment with actual decode execution. Experiments on production traces and large-scale simulations show that SIBYL reduces tail latency by up to 78% compared to state-of-the-art schedulers.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in llm inference with sarathi-serve. *Proceedings of 18th USENIX Symposium on Operating Systems Design and Implementation, 2024, Santa Clara*, 2024.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

DeepSeek-AI. Eplb: Expert parallel load balancing. https://github.com/deepseek-ai/EPLB, 2024. GitHub repository.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Doucet, Z., Sharma, R., de Vos, M., Pires, R., Kermarrec, A.-M., and Balmau, O. Harmoeny: Efficient multi-gpu inference of moe models, 2025. URL https://arxiv.org/abs/2506.12417.

Du, K., Wang, B., Zhang, C., Cheng, Y., Lan, Q., Sang, H., Cheng, Y., Yao, J., Liu, X., Qiao, Y., Stoica, I., and Jiang, J. Prefillonly: An inference engine for prefill-only workloads in large language model applications, 2025. URL https://arxiv.org/abs/2505.07203.

Fu, Y., Zhu, S., Su, R., Qiao, A., Stoica, I., and Zhang, H. Efficient llm scheduling by learning to rank, 2024. URL https://arxiv.org/abs/2408.15792.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., and Ma, Z. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Heisler, M., Yousefijamarani, Z., Wang, X., Wang, Q., Shi, G., Sadri, H., Yu, T., Li, Y., Li, H., Singh, G., et al. Llm inference scheduling: A survey of techniques, frameworks, and trade-offs. *Authorea Preprints*, 2025.

Jain, K., Parayil, A., Mallick, A., Choukse, E., Qin, X., Zhang, J., Goiri, I. n., Wang, R., Bansal, C., Rühle, V., Kulkarni, A., Kofsky, S., and Rajmohan, S. Performance aware llm load balancer for mixed workloads. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, EuroMLSys '25, pp. 19–30, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400715389. doi: 10.1145/3721146.3721947. URL https://doi.org/10.1145/3721146.3721947.

Jain, K., Parayil, A., Mallick, A., Choukse, E., Qin, X., Zhang, J., Íñigo Goiri, Wang, R., Bansal, C., Rühle, V., Kulkarni, A., Kofsky, S., and Rajmohan, S. Intelligent router for llm workloads: Improving performance through workload-aware load balancing, 2025b. URL https://arxiv.org/abs/2408.13510.

Jin, Y., Wu, C.-F., Brooks, D., and Wei, G.-Y. S^3: Increasing gpu utilization during generative inference for higher throughput, 2023. URL https://arxiv.org/abs/2306.06000.

Jin, Y., Wang, T., Lin, H., Song, M., Li, P., Ma, Y., Shan, Y., Yuan, Z., Li, C., Sun, Y., Wu, T., Chu, X., Huan, R., Ma, L., You, X., Zhou, W., Ye, Y., Liu, W., Xu, X., Zhang, Y., Dong, T., Zhu, J., Wang, Z., Ju, X., Song, J., Cheng, H., Li, X., Ding, J., Guo, H., and Zhang, Z. P/d-serve: Serving disaggregated large language model at scale, 2024. URL https://arxiv.org/abs/2408.08147.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Li, W., Jiang, G., Ding, X., Tao, Z., Hao, C., Xu, C., Zhang, Y., and Wang, H. Flowkv: A disaggregated inference framework with low-latency kv cache transfer and load-aware scheduling, 2025. URL https://arxiv.org/abs/2504.03775.

Lou, C., Qi, S., Jin, C., Nie, D., Yang, H., Ding, Y., Liu, X., and Jin, X. Hydraserve: Minimizing cold start latency for serverless llm serving in public clouds, 2025. URL https://arxiv.org/abs/2502.15524.

Ma, J., Chen, J., Li, X., Duan, J., Duanmu, H., Zhang, X., Yang, C., and Lin, D. *Tropical: Enhancing SLO Attainment in Disaggregated LLM Serving via SLO-Aware Multiplexing*. IEEE Press, 2025. ISBN 9798331503048. URL https://doi.org/10.1109/DAC63849.2025.11132617.

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., and Gao, J. Large language models: A survey, 2025. URL https://arxiv.org/abs/2402.06196.

MiniMax, Li, A., Gong, B., Yang, B., Shan, B., Liu, C., Zhu, C., Zhang, C., Guo, C., Chen, D., Li, D., Jiao, E., Li, G., Zhang, G., Sun, H., Dong, H., Zhu, J., Zhuang, J., Song, J., Zhu, J., Han, J., Li, J., Xie, J., Xu, J., Yan, J., Zhang, K., Xiao, K., Kang, K., Han, L., Wang, L., Yu, L., Feng, L., Zheng, L., Chai, L., Xing, L., Ju, M., Chi, M., Zhang, M., Huang, P., Niu, P., Li, P., Zhao, P., Yang, Q., Xu, Q., Wang, Q., Wang, Q., Li, Q., Leng, R., Shi, S., Yu, S., Li, S., Zhu, S., Huang, T., Liang, T., Sun, W., Sun, W., Cheng, W., Li, W., Song, X., Su, X., Han, X., Zhang, X., Hou, X., Min, X., Zou, X., Shen, X., Gong, Y., Zhu, Y., Zhou, Y., Zhong, Y., Hu, Y., Fan, Y., Yu, Y., Yang, Y., Li, Y., Huang, Y., Li, Y., Huang, Y., Xu, Y., Mao, Y., Li, Z., Li, Z., Tao, Z., Ying, Z., Cong, Z., Qin, Z., Fan, Z., Yu, Z., Jiang, Z., and Wu, Z. Minimax-01: Scaling foundation models with lightning attention, 2025. URL https://arxiv.org/abs/2501.08313.

NVIDIA. Dynamo: Designing LLM inference clusters for performance and energy efficiency. https://github.com/ai-dynamo/dynamo, 2025.

NVIDIA Corporation. NVIDIA Collective Communication Library (NCCL). https://github.com/NVIDIA/nccl, 2026.

OpenAI, :, Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., Iftimie, A., Karpenko, A., Passos, A. T., Neitz, A., Prokofiev, A., Wei, A., Tam, A., Bennett, A., Kumar, A., Saraiva, A., Vallone, A., Duberstein, A., Kondrich, A., Mishchenko, A., Applebaum, A., Jiang, A., Nair, A., Zoph, B., Ghorbani, B., Rossen, B., Sokolowsky, B., Barak, B., McGrew, B., Minaiev, B., Hao, B., Baker, B., Houghton, B., McKinzie, B., Eastman, B., Lugaresi, C., Bassin, C., Hudson, C., Li, C. M., de Bourcy, C., Voss, C., Shen, C., Zhang, C., Koch, C., Orsinger, C., Hesse, C., Fischer, C., Chan, C., Roberts, D., Kappler, D., Levy, D., Selsam, D., Dohan, D., Farhi, D., Mely, D., Robinson, D., Tsipras, D., Li, D., Oprica, D., Freeman, E., Zhang, E., Wong, E., Proehl, E., Cheung, E., Mitchell, E., Wallace, E., Ritter, E., Mays, E., Wang, F., Such, F. P., Raso, F., Leoni, F., Tsimpourlas, F., Song, F., von Lohmann, F., Sulit, F., Salmon, G., Parascandolo, G., Chabot, G., Zhao, G., Brockman, G., Leclerc, G., Salman, H., Bao, H., Sheng, H., Andrin, H., Bagherinezhad, H., Ren, H., Lightman, H., Chung, H. W., Kivlichan, I., O'Connell, I., Osband, I., Gilaberte, I. C., Akkaya, I., Kostrikov, I., Sutskever, I., Kofman, I., Pachocki, J., Lennon, J., Wei, J., Harb, J., Twore, J., Feng, J., Yu, J., Weng, J., Tang, J., Yu, J., Candela, J. Q., Palermo, J., Parish, J., Heidecke, J., Hallman, J., Rizzo, J., Gordon, J., Uesato, J., Ward, J., Huizinga, J., Wang, J., Chen, K., Xiao, K., Singhal, K., Nguyen, K., Cobbe, K., Shi, K., Wood, K., Rimbach, K., Gu-Lemberg, K., Liu, K., Lu, K., Stone, K., Yu, K., Ahmad, L., Yang, L., Liu, L., Maksin, L., Ho, L., Fedus, L., Weng, L., Li, L., McCallum, L., Held, L., Kuhn, L., Kondraciuk, L., Kaiser, L., Metz, L., Boyd, M., Trebacz, M., Joglekar, M., Chen, M., Tintor, M., Meyer, M., Jones, M., Kaufer, M., Schwarzer, M., Shah, M., Yatbaz, M., Guan, M. Y., Xu, M., Yan, M., Glaese, M., Chen, M., Lampe, M., Malek, M., Wang, M., Fradin, M., McClay, M., Pavlov, M., Wang, M., Wang, M., Murati, M., Bavarian, M., Rohaninejad, M., McAleese, N., Chowdhury, N., Chowdhury, N., Ryder, N., Tezak, N., Brown, N., Nachum, O., Boiko, O., Murk, O., Watkins, O., Chao, P., Ashbourne, P., Izmailov, P., Zhokhov, P., Dias, R., Arora, R., Lin, R., Lopes, R. G., Gaon, R., Miyara, R., Leike, R., Hwang, R., Garg, R., Brown, R., James, R., Shu, R., Cheu, R., Greene, R., Jain, S., Altman, S., Toizer, S., Toyer, S., Miserendino, S., Agarwal, S., Hernandez, S., Baker, S., McKinney, S., Yan, S., Zhao, S., Hu, S., Santurkar, S., Chaudhuri, S. R., Zhang, S., Fu, S., Papay, S., Lin, S., Balaji, S., Sanjeev, S., Sidor, S., Broda, T., Clark, A., Wang, T., Gordon, T., Sanders, T., Patwardhan, T., Sottiaux, T., Degry, T., Dimson, T., Zheng, T., Garipov, T., Stasi, T., Bansal, T., Creech, T., Peterson, T., Eloundou, T., Qi, V., Kosaraju, V., Monaco, V., Pong, V., Fomenko, V., Zheng, W., Zhou, W., McCabe, W., Zaremba, W., Dubois, Y., Lu, Y., Chen, Y., Cha, Y., Bai, Y., He, Y., Zhang, Y., Wang, Y., Shao, Z., and Li, Z. Openai o1 system card, 2024a. URL https://arxiv.org/abs/2412.16720.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. Gpt-4 technical report, 2024b. URL https://arxiv.org/abs/2303.08774.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, I. n., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, pp. 118–132. IEEE Press, 2025. ISBN 9798350326581. doi: 10.1109/ISCA59077.2024.00019. URL https://doi.org/10.1109/ISCA59077.2024.00019.

Qin, R., Li, Z., He, W., Zhang, M., Wu, Y., Zheng, W., and Xu, X. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2025. URL https://arxiv.org/abs/2407.00079.

Qwen Team. Qwen3-32b. ModelScope, 2025. URL https://www.modelscope.cn/models/Qwen/Qwen3-32B. Accessed: 2025-05-20.

SGLang. Sglang: A fast serving framework for large language models and vision-language models, 2025. URL https://github.com/sgl-project/sglang. Open-source under Apache-2.0 license.

ShareGPT Teams. ShareGPT. https://sharegpt.com/, 2023.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020. URL https://arxiv.org/abs/1909.08053.

Singh, G., Wang, X., Hu, Y., Yu, T., Xing, L., Jiang, W., Wang, Z., Bai, X., Li, Y., Xiong, Y., Zhang, Y., and Fan, Z. Efficiently serving large multimodal models using epd disaggregation, 2025. URL https://arxiv.org/abs/2501.05460.

Sun, B., Huang, Z., Zhao, H., Xiao, W., Zhang, X., Li, Y., and Lin, W. Llumnix: Dynamic scheduling for large language model serving, 2024. URL https://arxiv.org/abs/2406.03243.

Team, ., Zeng, A., Lv, X., Zheng, Q., Hou, Z., Chen, B., Xie, C., Wang, C., Yin, D., Zeng, H., Zhang, J., Wang, K., Zhong, L., Liu, M., Lu, R., Cao, S., Zhang, X., Huang, X., Wei, Y., Cheng, Y., An, Y., Niu, Y., Wen, Y., Bai, Y., Du, Z., Wang, Z., Zhu, Z., Zhang, B., Wen, B., Wu, B., Xu, B., Huang, C., Zhao, C., Cai, C., Yu, C., Li, C., Ge, C., Huang, C., Zhang, C., Xu, C., Zhu, C., Li, C., Yin, C., Lin, D., Yang, D., Jiang, D., Ai, D., Zhu, E., Wang, F., Pan, G., Wang, G., Sun, H., Li, H., Li, H., Hu, H., Zhang, H., Peng, H., Tai, H., Zhang, H., Wang, H., Yang, H., Liu, H., Zhao, H., Liu, H., Yan, H., Liu, H., Chen, H., Li, J., Zhao, J., Ren, J., Jiao, J., Zhao, J., Yan, J., Wang, J., Gui, J., Zhao, J., Liu, J., Li, J., Li, J., Lu, J., Wang, J., Yuan, J., Li, J., Du, J., Du, J., Liu, J., Zhi, J., Gao, J., Wang, K., Yang, L., Xu, L., Fan, L., Wu, L., Ding, L., Wang, L., Zhang, M., Li, M., Xu, M., Zhao, M., Zhai, M., Du, P., Dong, Q., Lei, S., Tu, S., Yang, S., Lu, S., Li, S., Li, S., Shuang-Li, Yang, S., Yi, S., Yu, T., Tian, W., Wang, W., Yu, W., Tam, W. L., Liang, W., Liu, W., Wang, X., Jia, X., Gu, X., Ling, X., Wang, X., Fan, X., Pan, X., Zhang, X., Zhang, X., Fu, X., Zhang, X., Xu, Y., Wu, Y., Lu, Y., Wang, Y., Zhou, Y., Pan, Y., Zhang, Y., Wang, Y., Li, Y., Su, Y., Geng, Y., Zhu, Y., Yang, Y., Li, Y., Wu, Y., Li, Y., Liu, Y., Wang, Y., Li, Y., Zhang, Y., Liu, Z., Yang, Z., Zhou, Z., Qiao, Z., Feng, Z., Liu, Z., Zhang, Z., Wang, Z., Yao, Z., Wang, Z., Liu, Z., Chai, Z., Li, Z., Zhao, Z., Chen, W., Zhai, J., Xu, B., Huang, M., Wang, H., Li, J., Dong, Y., and Tang, J. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025a. URL https://arxiv.org/abs/2508.06471.

Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., Chen, Z., Cui, J., Ding, H., Dong, M., Du, A., Du, C., Du, D., Du, Y., Fan, Y., Feng, Y., Fu, K., Gao, B., Gao, H., Gao, P., Gao, T., Gu, X., Guan, L., Guo, H., Guo, J., Hu, H., Hao, X., He, T., He, W., He, W., Hong, C., Hu, Y., Hu, Z., Huang, W., Huang, Z., Huang, Z., Jiang, T., Jiang, Z., Jin, X., Kang, Y., Lai, G., Li, C., Li, F., Li, H., Li, M., Li, W., Li, Y., Li, Y., Li, Z., Li, Z., Lin, H., Lin, X., Lin, Z., Liu, C., Liu, C., Liu, H., Liu, J., Liu, J., Liu, L., Liu, S., Liu, T. Y., Liu, T., Liu, W., Liu, Y., Liu, Y., Liu, Y., Liu, Y., Liu, Z., Lu, E., Lu, L., Ma, S., Ma, X., Ma, Y., Mao, S., Mei, J., Men, X., Miao, Y., Pan, S., Peng, Y., Qin, R., Qu, B., Shang, Z., Shi, L., Shi, S., Song, F., Su, J., Su, Z., Sun, X., Sung, F., Tang, H., Tao, J., Teng, Q., Wang, C., Wang, D., Wang, F., Wang, H., Wang, J., Wang, J., Wang, J., Wang, S., Wang, S., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Y., Wang, Z., Wang, Z., Wang, Z., Wei, C., Wei, Q., Wu, W., Wu, X., Wu, Y., Xiao, C., Xie, X., Xiong, W., Xu, B., Xu, J., Xu, J., Xu, L. H., Xu, L., Xu, S., Xu, W., Xu, X., Xu, Y., Xu, Z., Yan, J., Yan, Y., Yang, X., Yang, Y., Yang, Z., Yang, Z., Yang, Z., Yao, H., Yao, X., Ye, W., Ye, Z., Yin, B., Yu, L., Yuan, E., Yuan, H., Yuan, M., Zhan, H., Zhang, D., Zhang, H., Zhang, W., Zhang, X., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Y., Zhang, Z., Zhao, H., Zhao, Y., Zheng, H., Zheng, S., Zhou, J., Zhou, X., Zhou, Z., Zhu, Z., Zhuang, W., and Zu, X. Kimi k2: Open agentic intelligence, 2025b. URL https://arxiv.org/abs/2507.20534.

Team, Q. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Wang, Z., Hong, Z., Li, X., Wang, Z., Li, S., Meng, Q., Wang, Q., Huan, C., Gu, R., Zhong, S., and Tian, C. Adaptive rescheduling in prefill-decode disaggregated llm inference, 2025. URL https://arxiv.org/abs/2510.13668.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Wu, B., Zhong, Y., Zhang, Z., Liu, S., Liu, F., Sun, Y., Huang, G., Liu, X., and Jin, X. Fast distributed inference serving for large language models, 2024. URL https://arxiv.org/abs/2305.05920.

Xiang, Y., Li, X., Qian, K., Yang, Y., Zhu, D., Yu, W., Zhai, E., Liu, X., Jin, X., and Zhou, J. Aegaeon: Effective gpu pooling for concurrent llm serving on the market. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 1030–1045, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764815. URL https://doi.org/10.1145/3731569.3764815.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zed Industries. Zeta: A dataset for predictive code editing. Hugging Face Datasets, 2024. URL https://huggingface.co/datasets/zed-industries/zeta. Accessed: 2025-12-02.

Zhang, Y., Chen, Y., Mo, X., Xi, A., Li, J., and Wu, W. A predictive and synergistic two-layer scheduling framework for llm serving, 2025. URL https://arxiv.org/abs/2509.23384.

Zhao, C., Wu, W., Song, L., Xu, Y., and Yuan, Y. Fine-grained moe load balancing with linear programming, 2026. URL https://arxiv.org/abs/2511.16947.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., and Wen, J.-R. A survey of large language models, 2025. URL https://arxiv.org/abs/2303.18223.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024.

Table of Contents

A. Why Tail Latency Matters?

In production LLM serving, relying solely on central tendency metrics (such as Mean or P50) often creates a false sense of security regarding system performance. Figure 10 illustrates a typical latency distribution observed under standard scheduling policies. While the median (P50) TPOT comfortably satisfies the stringent SLO of 100ms (indicated by the red dashed line), the distribution exhibits a severe long-tail phenomenon. As shown, the tail latencies degrade exponentially: the P99 latency exceeds 1500ms, and the P99.9 latency spikes to over 2500ms, more than **25× higher** than the SLO threshold. For interactive applications like chatbots, these tail latencies manifest as noticeable "stuttering" during text generation, severely disrupting the user experience. Consequently, optimizing for tail latency (P99/P99.9) rather than P50 is critical for ensuring consistent quality of service, motivating the design of SIBYL to specifically mitigate these worst-case scenarios.

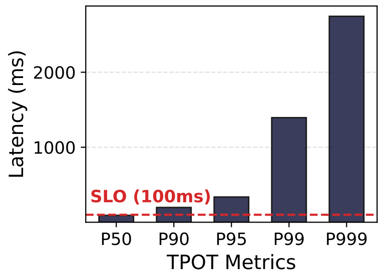


Figure 10. **The long-tail latency problem.** *Although the median (P50) TPOT meets the 100ms SLO (red dashed line), severe congestion causes P99 and P99.9 latencies to violate the constraint by orders of magnitude, highlighting the inadequacy of P50 as a sole metric.*

B. Summary of Notations

To facilitate a better understanding of the formulation presented in Section 4, we summarize the key mathematical notations and system variables used in SIBYL. Table 2 categorizes these symbols by their distinct roles.

Table 2. **Key notations.** *Symbols are categorized by their roles in the probabilistic workload modeling.*

Symbol	**Description**
System State & Scheduling	
t_{now}	Current timestamp when the scheduling decision is made
τ	Projected handoff moment (t_{now} + prefill_duration)
$\mathcal{D}, j$	Set of decode instances and a candidate instance $j \in \mathcal{D}$
$Q_j^{\text{run}}, Q_j^{\text{pend}}$	Sets of currently active tasks and pending prefill tasks on j
$v_k, \bar{v}_{\text{sys}}$	Individual decoding speed (Type 1) and system average speed
Probabilistic Primitives	
L, l_k^{in}	Random variable for generation length; input prompt length
$l_k(t)$	Generated sequence length of task k at time t
$\mathcal{M}_k(\tau)$	**Physical Pressure**: Token occupancy ($l_k^{\text{in}} + l_k(\tau)$) at τ
$S(x), \hat{S}(x)$	Theoretical survival function and its online estimator
$\mathcal{P}_k(\tau)$	**Cond. Survival Prob.**: Likelihood that task k persists to τ
Speculative Load Composition	
$\mathcal{L}_j(\tau)$	Total projected load metric for instance j (to minimize)
$\mathcal{L}_{\text{I}}(k, \tau)$	Projected load from **Active Tasks** (Type 1)
$\mathcal{L}_{\text{II}}(k, \tau)$	Projected load from **Pre-Handoff Shadow Tasks** (Type 2)
$\mathcal{L}_{\text{III}}(k, \tau)$	Interference penalty from **Post-Handoff Tasks** (Type 3)

C. Expected Load at Time τ

We define the expected system load $\mathbb{L}$ at time τ as the aggregate decoding load contributed by all tasks that are active at this moment:

$$\mathbb{L}(\tau) = \sum_k \mathcal{M}_k(\tau)\, \mathbb{I}_{(\tau_k \le \tau,\, T_k > \tau)}, \tag{9}$$

where $\mathcal{M}_k(\tau)$ denotes the decoding load of task k at time τ, τ_k is its decode start time, and T_k is its completion time. The indicator function ensures that only tasks that have started decoding but have not yet completed contribute to the system load.

We take expectation with respect to the stochastic task lifetimes, conditioned on the system state observed at the current time t_0. Let $\mathbb{E}_0[\cdot]$ denote expectation under this conditioning. Then,

$$\mathbb{E}_0[\mathbb{L}(\tau)] = \sum_k \mathbb{E}_0\left[\mathcal{M}_k(\tau)\, \mathbb{I}_{(\tau_k \le \tau,\, T_k > \tau)}\right]. \tag{10}$$

For an active decoding task, its load evolves linearly over time. Specifically, for any future time $\tau \ge t_0$, we model the decoding load as

$$\mathcal{M}_k(\tau) = l_{k,0} + v_k \cdot (\tau - t_0), \tag{11}$$

where $l_{k,0}$ is the accumulated decoding load at time t_0 and v_k is the decoding rate. Given the system state at t_0, both quantities are known.

Substituting (11) into (10) and noting that $\mathcal{M}_k(\tau)$ is measurable with respect to the information available at time t_0, we can factor it out of the conditional expectation:

$$\begin{aligned}\mathbb{E}_0[\mathbb{L}(\tau)] &= \sum_k \mathbb{E}_0\left[\big(l_{k,0} + v_k \cdot (\tau - t_0)\big)\mathbb{I}_{(\tau_k \le \tau,\, T_k > \tau)}\right] \\ &= \sum_k \big(l_{k,0} + v_k \cdot (\tau - t_0)\big)\mathbb{P}_0(T_k > \tau \mid \tau_k \le \tau)\,. \end{aligned} \tag{12}$$

The remaining uncertainty lies entirely in whether a task survives until time τ. This survival probability can be equivalently expressed in terms of the residual decoding length:

$$\mathbb{P}_0(T_k > \tau) = \mathbb{P}_0(l_k(\tau_k) > l_k(\tau)) = \mathbb{P}_0(L_k > l_k(\tau))\,, \tag{13}$$

where L_k denotes the total decoding length of task k. This probability depends only on the residual decoding length distribution conditioned on the current system state.

Putting everything together, the expected system load at time τ admits the following factorized form:

$$\mathbb{E}_0[\mathbb{L}(\tau)] = \sum_k \underbrace{\big(l_{k,0} + v_k \cdot (\tau - t_0)\big)}_{\mathcal{M}_k(\tau)} \underbrace{\mathbb{P}_0(L_k > l_k(\tau))}_{\mathcal{P}_k(\tau)}\,. \tag{14}$$

This expression represents the expected contribution of each task as its projected decoding load weighted by the probability of remaining active at time τ.

For Type 1 and Type 2 tasks, the formulation differs only in the reference state used for projection, *i.e.*, the choice of t_0, the accumulated load $l_{k,0}$ at that time, and the decoding rate v_k; the expected-load expression in (14) remains identical.

D. Temporal Consistency

Our formulation is designed to be *boundary-consistent* across state transitions to ensure scheduling stability. First, at the *Active-Shadow boundary* (Type 1 $\leftrightarrow$ Type 2), a task transitioning from queue to execution shifts from using the system-average speed ($\bar{v}_{sys}$) to its specific rate (v_k). Mathematically, since v_k is initially unknown, $\bar{v}_{sys}$ serves as its unbiased estimator at the boundary ($t = 0$). While the realization of v_k during execution may introduce a value update, the *expected load* remains continuous at the instant of transition.

Second, at the *Shadow-Decay boundary* (Type 2 $\leftrightarrow$ Type 3), the model naturally converges: as the projected start time aligns with the handoff ($\tau_k \to \tau$), the generation component vanishes, and both Type 2 and Type 3 models resolve identically to the intrinsic input cost ($\lim_{\tau_k \to \tau} \mathcal{L} = l_k^{in}$). This design eliminates singularity points in the objective function, allowing the scheduler to smoothly track tasks as they drift through different lifecycle stages.

E. Implementation of SIBYL

SIBYL is implemented as a lightweight, non-intrusive middleware that operates entirely in the control plane. This framework-agnostic design allows SIBYL to be deployed alongside mainstream inference engines, such as vLLM and SGLang, without requiring invasive modifications to the underlying kernels or model executors.

E.1. Standalone Architecture

Similar to the reference implementation in vLLM's disaggregated serving example,[2] SIBYL functions as a standalone HTTP proxy. This decoupled architecture allows SIBYL to manage request routing and scheduling decisions centrally while maintaining compatibility with standard OpenAI-compatible API interfaces.

E.2. State Synchronization

To perform speculative scheduling, SIBYL maintains a real-time view of the cluster state. Instead of relying on complex distributed consensus protocols, SIBYL utilizes the native monitoring interfaces provided by vLLM. Specifically, the proxy periodically polls the `/metrics` endpoint of each decode instance via HTTP `GET` requests. By parsing these metrics—which include the number of running requests, KV cache usage, and iteration stats—SIBYL updates its internal probabilistic workload model with negligible overhead.

E.3. Enabling Reasoning in vLLM Benchmarks

Standard benchmarking tools (*e.g.,* `vllm bench`) often lack native support for passing engine-specific parameters required by reasoning models. To evaluate reasoning capabilities (e.g., Chain-of-Thought) systematically, we implemented a request injection mechanism within the SIBYL proxy. For every incoming inference request, the proxy intercepts the JSON payload and automatically injects the necessary configuration to trigger the model's reasoning mode. Specifically, we modify the request body to include:

```
request_data['extra_body']['chat_template_kwargs'] = {'enable_thinking': True}
```

This ensures that all requests processed during our evaluation—regardless of the client-side configuration—correctly trigger the reasoning logic in the reasoning model.

[2]Specifically, `disagg_proxy_p2p_nccl_xpyd.py`.

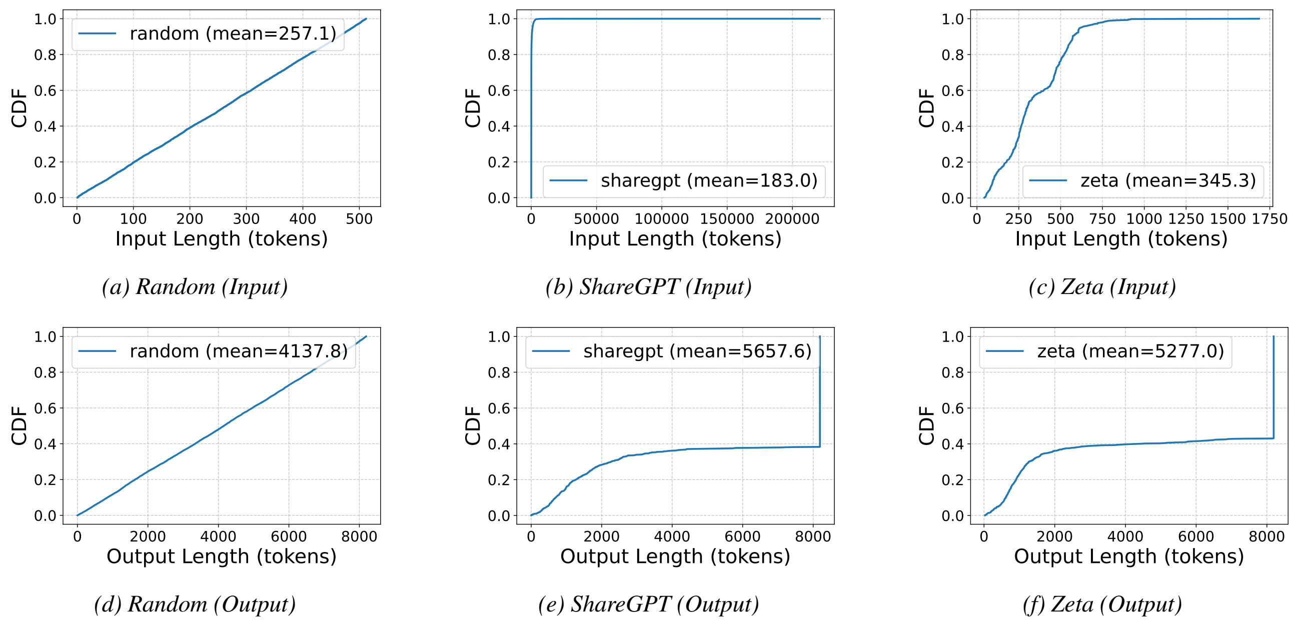
F. Detailed Experimental Setup

F.1. Model Configurations

To enable the reasoning capabilities of Qwen3-32B and avoid deterministic degeneration, we disable greedy decoding and employ stochastic sampling. We adopt the official hyperparameters recommended by Qwen Team (2025), as detailed in Table 3.

Table 3. Generation hyperparameters used to enable reasoning mode.

Parameter	Value
Temperature	0.6
Top-P	0.95
Top-K	20
Min-P	0
Greedy Decoding	Disabled



(a) Random (Input) *(b) ShareGPT (Input)* *(c) Zeta (Input)*

(d) Random (Output) *(e) ShareGPT (Output)* *(f) Zeta (Output)*

Figure 11. **Cumulative Distribution Function (CDF) of token lengths.** *The top row shows the input prompt distribution, while the bottom row illustrates the output generation length distribution across Random, ShareGPT, and Zeta datasets.*

F.2. Dataset Details

We evaluate our system using three datasets that cover synthetic micro-benchmarks, real-world conversations, and code generation tasks. The input and output length distributions are visualized in Figure 11.

Random. This synthetic dataset is designed to micro-benchmark system performance under reasoning-intensive scenarios (e.g., Chain-of-Thought), where short questions trigger long responses. We uniformly sample input prompt lengths from $[1, 512]$ and output lengths from $[1, 8192]$. This configuration stresses the decode phase and allows us to test the scheduler's behavior under uniform workload variance.

ShareGPT. Representing real-world human-chatbot interactions, this dataset features relatively short prompts (mean length 183) but exhibits high variance in response lengths. Under our reasoning-enabled configuration, the output distribution becomes extremely heavy-tailed; as shown in Figure 11(e), a significant portion of requests saturates the maximum token limit (8192), creating sustained pressure on decode instances.

Zeta. Sourced from predictive code editing tasks, this dataset introduces longer context prefixes compared to ShareGPT, with input lengths extending up to 1.7k tokens (mean 345). Similar to ShareGPT, the output distribution is dominated by long-running tasks that frequently hit the generation cap, effectively testing the system's ability to handle heavier prefill loads alongside concurrent long-context decoding.

```
# Execution Command for Prefill Instance
vllm serve <MODEL_PATH> \
   --host <HOST_IP> \
   --port <HTTP_PORT> \
   --served-model-name qwen3-32b \
   --tensor-parallel-size 1 \
   --seed 1024 \
   --dtype float16 \
   --max-model-len 32768 \
   --max-num-batched-tokens 32768 \
   --max-num-seqs 1 \
   --trust-remote-code \
   --reasoning-parser qwen3 \
   --gpu-memory-utilization 0.9 \
   --compilation-config '{"cudagraph_mode": "PIECEWISE"}' \
   --kv-transfer-config '{
      "kv_connector": "P2pNcclConnector",
      "kv_role": "kv_producer",
      "kv_buffer_size": "1e1",
      "kv_port": "<KV_PORT>",
      "kv_connector_extra_config": {
         "proxy_ip": "<PROXY_IP>",
         "proxy_port": "<PROXY_PORT>",
         "http_port": "<HTTP_PORT>"
      }
   }'
```

```
# Execution Command for Decode Instance
vllm serve <MODEL_PATH> \
   --host <HOST_IP> \
   --port <HTTP_PORT> \
   --served-model-name qwen3-32b \
   --tensor-parallel-size 1 \
   --seed 1024 \
   --dtype float16 \
   --max-model-len 32768 \
   --max-num-batched-tokens 32768 \
   --max-num-seqs 256 \
   --trust-remote-code \
   --reasoning-parser qwen3 \
   --gpu-memory-utilization 0.8 \
   --kv-transfer-config '{
      "kv_connector": "P2pNcclConnector",
      "kv_role": "kv_consumer",
      "kv_buffer_size": "8e9",
      "kv_port": "<KV_PORT>",
      "kv_connector_extra_config": {
         "proxy_ip": "<PROXY_IP>",
         "proxy_port": "<PROXY_PORT>",
         "http_port": "<HTTP_PORT>"
      }
   }'
```

```
# Benchmarking Command
vllm bench serve \
   --backend vllm \
   --model qwen3-32b \
   --tokenizer <MODEL_PATH> \
   --dataset-name "hf" \
   --dataset-path <DATASET_PATH> \
   --hf-name <DATASET_NAME> \
   --host <PROXY_IP> \
   --port <PROXY_PORT> \
   --burstiness <BURSTINESS> \
   --percentile-metrics "ttft,tpot,itl,e2el" \
   --metric-percentiles "50,90,95,99,99.9" \
   --seed $(date +%s) \
   --trust-remote-code \
   --request-rate <RPS> \
   --num-prompts <NUM_PROMPTS>
```

F.3. vLLM Runtime Configuration

We deploy the disaggregated inference system using vLLM. Specifically, we configure the system with the `P2pNcclConnector`, which natively supports the overlapping of prefill computation with KV cache transmission. The following listings provide the representative command-line templates used in our experiments. Environment-specific variables (*e.g.,* IPs, ports) and dynamic workload parameters have been replaced with placeholders for generality.

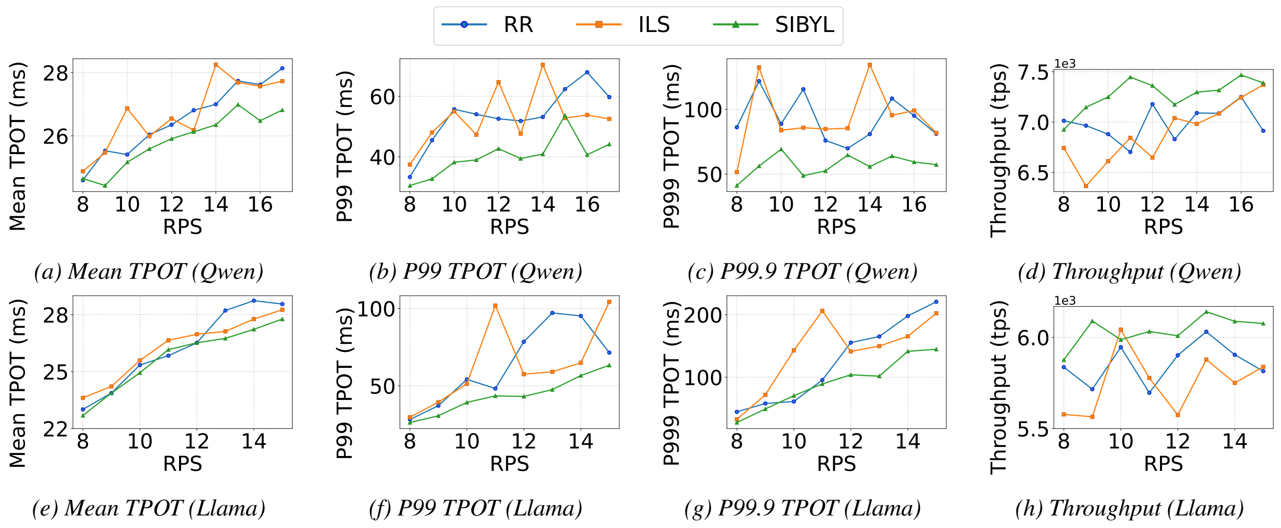


Figure 12. End-to-end performance comparison of Qwen2.5-7B-Instruct (Team, 2024) and Llama3-8B (AI@Meta, 2024) on random dataset.

G. Additional Experiment Results

The primary evaluation in the main text utilizes NVIDIA H20 GPUs. As detailed in Table 4, the H20 features massive HBM capacity (141 GB) and high bandwidth (4.0 TB/s) but relatively lower compute power (148 TFLOPS, BF16), making them exceptionally suitable for the memory-bound decode phase of LLM inference. To validate the broad applicability and robustness of SIBYL across diverse hardware architectures, this section supplements those results with experiments on the NVIDIA A100—a flagship predecessor characterized by significantly higher compute capability (312 TFLOPS, BF16) but relatively more constrained memory resources (40GB HBM, 1.89 TB/s).

Table 4. Hardware specification comparison between NVIDIA H20 and A100 GPUs.

Specification	**NVIDIA H20**	**NVIDIA A100**
FP16/BF16 Compute	148 TFLOPS	312 TFLOPS
HBM Capacity	141 GB	40 GB
HBM Bandwidth	4.0 TB/s	1.89 TB/s

G.1. Experimental Setup

Testbed. We also conduct our experiments on a distributed cluster consisting of two physical servers equipped with NVIDIA A100 GPUs. To simulate a disaggregated serving environment, we deploy the system in a 2P4D topology (2 prefill instances and 4 decode instances). Specifically, the prefill workload is handled by a server with 2×A100 GPUs, while the decode workload is hosted on a separate server with 4×A100 GPUs. In this configuration, each GPU hosts a single model replica without tensor parallelism.

Model. We evaluate performance using two widely adopted open-weights models: Qwen2.5-7B-Instruct (Team, 2024) and Llama3-8B (AI@Meta, 2024).

Dataset. To micro-benchmark the system under controlled workload variance, we employ a synthetic random dataset. The requests are generated with input prompt lengths uniformly distributed in the range $[1, 1024]$ and output generation lengths uniformly distributed in $[1, 4096]$.

G.2. Overall Performance

Figure 12 illustrates the end-to-end performance comparison on the random dataset. We observe that SIBYL consistently outperforms both RR and ILS baselines across different models and varying request rates (RPS), particularly in suppressing tail latency and improving system throughput.

Qwen2.5-7B Results. As shown in the top row of Figure 12, SIBYL demonstrates robust stability under high contention. compared to ILS, SIBYL reduces the average P99 and P99.9 TPOT by **23.0%** and **36.2%**, respectively, with peak reductions reaching up to **58.6%** in the most congested scenarios. Similarly, against RR, SIBYL achieves average reductions of **24.2%** (P99) and **36.4%** (P99.9). In terms of throughput, SIBYL improves the average throughput by **5.7%** over ILS and **4.2%** over RR, effectively handling higher loads without violating latency constraints.

Llama3-8B Results. The bottom row of Figure 12 confirms the generalizability of our approach. SIBYL achieves significant tail latency improvements, reducing the average P99 TPOT by **26.4%** vs. ILS and **26.2%** vs. RR. For the strict P99.9 metric, SIBYL lowers the latency by an average of **32.0%** compared to ILS. Throughput gains remain consistent, with SIBYL outperforming ILS by an average of **5.1%** and RR by **3.1%**.

While the improvement in Mean TPOT is modest (approximately 2–3% across workloads), the substantial reduction in tail latency (P99/P99.9) indicates that SIBYL effectively mitigates the severe congestion instances caused by suboptimal scheduling. Crucially, this suppression of tail latency directly translates to significantly fewer SLO violations, thereby guaranteeing a higher quality of service.

H. Overhead Analysis

To assess the implementation cost of SIBYL, we evaluate the scheduling overhead in terms of decision latency and computational resource consumption on the proxy node.

Table 5. **Comparison of scheduling decision latency.** SIBYL *incurs a minor overhead due to prefill prediction but remains negligible compared to total request latency (i.e., end-to-end latency).*

Method	Decision Latency (ms)
RR	<1
ILS	<1
SIBYL	5.07

Decision Latency. Table 5 reports the average time required to dispatch a single request. While simple policies like RR and ILS operate instantly (< 1 ms), SIBYL introduces an average latency of 5.07 ms. The majority of this time is attributed to the prefill duration prediction model (used to estimate the handoff moment). However, given that end-to-end inference latencies for LLM workloads typically range from seconds to minutes, this millisecond-level overhead is negligible (orders of magnitude smaller) and does not impact the user experience.

Computational Cost. Since the scheduling logic for all methods resides on the CPU-based proxy, we monitor the CPU utilization to verify efficiency. As illustrated in Figure 13, SIBYL exhibits a CPU utilization profile highly similar to the baseline methods, fluctuating within a low range (2.5%–4.5%) without noticeable spikes. This demonstrates that the probabilistic calculations and state tracking in SIBYL are lightweight and do not impose a computational bottleneck on the serving system.

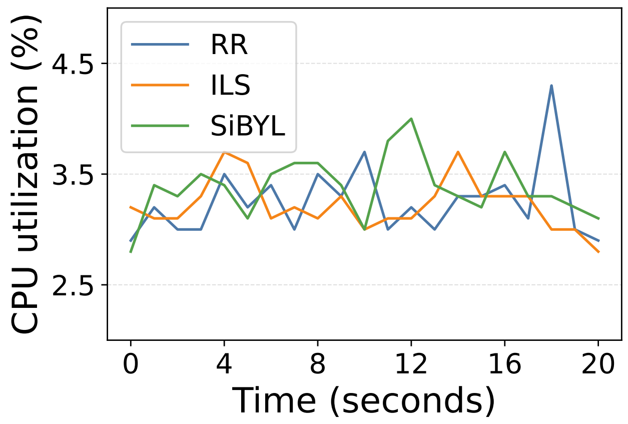


Figure 13. **CPU utilization of the scheduler proxy over time.** SIBYL *maintains a utilization profile comparable to lightweight baselines (RR/ILS), indicating high computational efficiency.*

I. Simulation Methodology

To evaluate SIBYL at scales exceeding typical physical testbeds (e.g., 64–256 decode instances) and to conduct extensive stress testing under varied arrival rates, we implemented a high-fidelity, trace-driven discrete-event simulator. This section details the simulation environment, the hardware modeling assumptions, and the implementation logic corresponding to Algorithms 2 and 3.

I.1. Hardware Modeling

Unlike basic simulators that assume constant processing speeds, our model explicitly accounts for memory bandwidth contention. Since LLM decoding is bandwidth-bound, we model the GPU as a shared resource where the total throughput is divided among active requests. Consequently, as the number of concurrent tasks increases, the execution speed of each individual request decreases, accurately reflecting the performance slowdown in real hardware.

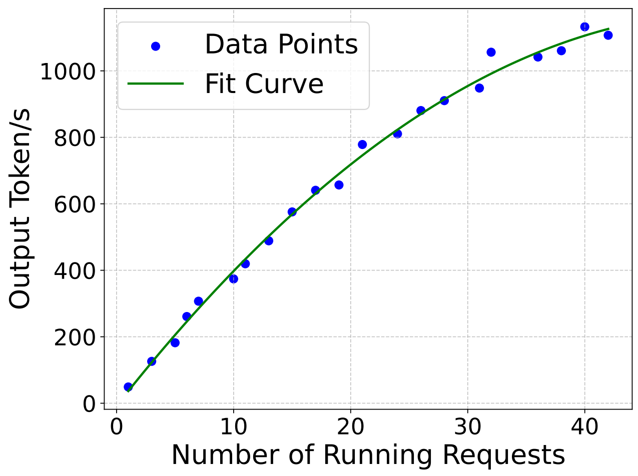


Figure 14. TPS profiling on NVIDIA H20 GPU.

Dynamic Throughput. The total system throughput (TPS_{total}) of a decode instance is not static; it varies non-linearly with the level of concurrency. Instead of relying on proxy metrics like KV cache occupancy, we directly model throughput as a function of the instantaneous batch size (number of running requests), denoted as N. As illustrated in Figure 14, we profile the NVIDIA H20 GPU and fit the observed performance to a quadratic polynomial:

$$TPS_{total}(N) = \alpha N^2 + \beta N + \gamma \tag{15}$$

Algorithm 2 Trace-Driven Discrete-Event Simulation Framework

```
Input   : Request Trace $\mathcal{T}$, Cluster Config $\mathcal{C}$, Scheduler $\mathcal{S}$
Output : Latency Metrics, Throughput
21 Initialize $N$ Decode instances, global RequestTable (for SIBYL), and set $t_{now} \leftarrow 0$
22 for each request $r$ in $\mathcal{T}$ sorted by arrival time do
23     $t_{now} \leftarrow r.arrival\_time$
24     for each instance $j \in \{1 \dots N\}$ do
25         ADVANCEINSTANCE($j, t_{now}$) ;                         // See Algorithm 3
26         Update instance speed $v_j$ based on dynamic load
27     if $\mathcal{S}$ is SIBYL then
28         Sync RequestTable with physical state
29         $\delta_{pre} \leftarrow$ EstimatePrefill($r.input\_len$)
30         $\tau \leftarrow t_{now} + \delta_{pre}$
31         $j^* \leftarrow \arg\min_j$ ProjectedLoad($j, \tau$)
32     else if $\mathcal{S}$ is ILS then
33         $j^* \leftarrow \arg\min_j$ CurrentLoad($j, t_{now}$)
34     else
35         $j^* \leftarrow (j_{last} + 1) \pmod N$
36     $t_{start\_decode} \leftarrow t_{now} + \delta_{pre}$
37     Push event START_DECODE($r$) to instance $j^*$ queue at $t_{start\_decode}$
38 Flush all instances until completion
```

where α, β, and γ are hardware-specific coefficients derived from regression analysis. For the specific hardware configuration used in our experiments, the fitted parameters are $\alpha \approx -0.423$, $\beta \approx 44.766$, and $\gamma \approx -7.753$. This model accurately captures the hardware's behavior, where throughput initially scales linearly with batch size before plateauing due to memory bandwidth saturation. In our simulation (Algorithm 3, Line 4), we recalculate the instance speed using this fitted function whenever the batch size N changes.

Bandwidth Contention. When multiple requests decode concurrently, they share the GPU's memory bandwidth. The simulator models this by splitting the total throughput equally among all active tasks (Algorithm 3, Line 5). Consequently, the effective speed v_{speed} of a single request decreases as the batch size increases, accurately simulating the slowdown effect caused by multiple parallel requests.

I.2. Simulation Logic and Workflow

The simulation proceeds as a strict discrete-event system driven by the request arrival trace. The interaction between the high-level scheduling logic and the low-level physics engine is detailed below:

1. Global Event Loop (Algorithm 2). The framework processes requests sequentially based on their arrival times. Upon each arrival at time t_{now}, the system first ensures state synchronization (Lines 6-9) by invoking the ADVANCEINSTANCE subroutine for every instance; this advances the physical simulation to the current timestamp, processing any interim task completions or new decoding starts. With the system state synchronized, the scheduler then executes the decision-making logic (Lines 11-17) to select a target instance j^*. For SIBYL, this involves syncing the shadow RequestTable and projecting loads at the future handoff time τ. Crucially, to explicitly model the temporal gap (Lines 19-20), the request is not executed immediately; instead, a START_DECODE event is registered in the target instance's queue, scheduled to trigger strictly after the prefill duration at $t_{now} + \delta_{pre}$.

2. Instance State Evolution (Algorithm 3). This subroutine acts as the physics engine. It advances the state of a specific instance continuously. Instead of fixed time steps, it calculates Δt_{step} (Line 9) as the minimum time until the next discrete event: either a running task finishes its generation (Δt_{finish}) or a pending prefill task enters the decode phase (Δt_{start}). This event-driven progression ensures that resource contention is recalculated exactly when the system load changes, guaranteeing high temporal precision.

Algorithm 3 Instance State Evolution (Processor Sharing Model)

```
Function AdvanceInstance($instance, t_{target}$)
    // Simulates continuous execution up to $t_{target}$
39  while $instance.time < t_{target}$ do
40      $Q_{run} \leftarrow instance.decoding\_tasks$
41      $L_{tokens} \leftarrow \sum_{k \in Q_{run}} (len_{prompt,k} + len_{generated,k})$
42      $TPS_{total} \leftarrow$ GetFittedThroughput($|Q_{run}|$) ;          // Based on profiling curve (Figure 14)
43      $v_{speed} \leftarrow TPS_{total}/|Q_{run}|$ ;                       // Processor sharing: split equally
        // Determine time to next state change
44      $\Delta t_{finish} \leftarrow \min_{k \in Q_{run}} (remaining\_tokens_k / v_{speed})$
45      $\Delta t_{start} \leftarrow instance.next\_start\_event - instance.time$
46      $\Delta t_{step} \leftarrow \min(\Delta t_{finish}, \Delta t_{start}, t_{target} - instance.time)$
        // Advance physical state
47      for each task $k$ in $Q_{run}$ do
48          $remaining\_tokens_k \leftarrow remaining\_tokens_k - (\Delta t_{step} \times v_{speed})$
49      $instance.time \leftarrow instance.time + \Delta t_{step}$
        // Handle discrete events
50      if $\Delta t_{step} == \Delta t_{finish}$ then
51          Identify task $k^*$ with $remaining \leq 0$
52          Remove $k^*$ from $Q_{run}$, record $t_{end}$ and TPOT
53      else if $\Delta t_{step} == \Delta t_{start}$ then
54          Pop task $r_{new}$ from event queue
55          Add $r_{new}$ to $Q_{run}$ with initial remaining load
```

I.3. Extended Simulation Results

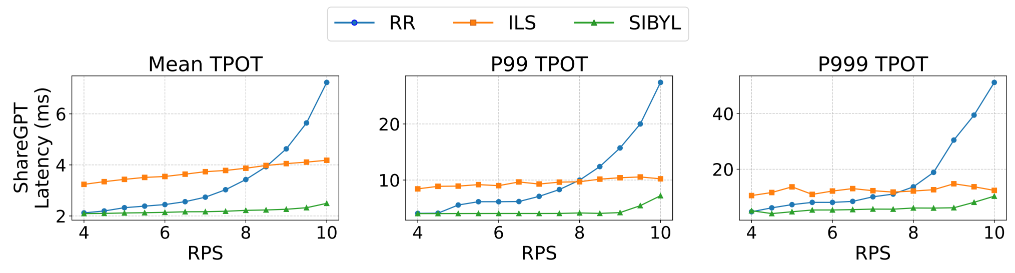
To verify the scalability of SIBYL in massive production environments, we extend our simulation to clusters with 128 and 256 decode instances. Figure 15 presents the latency characteristics under sweeping arrival rates.

Latency Stability at Scale. SIBYL demonstrates robust scalability, maintaining a flat latency profile even as the system approaches saturation. In contrast, the benchmark methods suffer from varying degrees of performance degradation. On the heavy-tailed ShareGPT workload, RR blindly routes long requests to busy instances, causing unbalanced load. SIBYL effectively suppresses this tail latency, outperforming RR by 44.1% (128 instances) and 31.6% (256 instances) in P99 TPOT, while surpassing ILS by 53.9% and 62.5% respectively. On the Zeta dataset, the gap widens further as ILS struggles with long prefill durations that create large temporal gaps. This "invisible load" issue renders ILS's instantaneous view obsolete, causing severe congestion. Consequently, SIBYL achieves drastic reductions in P99 TPOT against ILS: 71.5% on 128 instances and 82.9% on 256 instances, confirming that predictive lookahead is critical for scaling coding workloads.

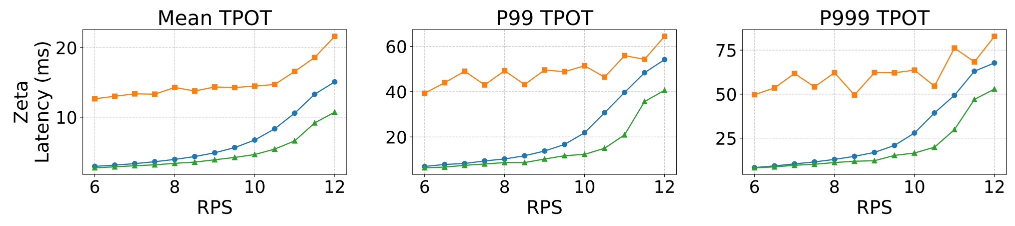
Mechanistic Explanation via Assignment Accuracy. Table 6 elucidates the root cause of these performance gaps. SIBYL achieves superior precision, maintaining $> 91\%$ accuracy on ShareGPT and significantly outperforming baselines on Zeta. Notably, ILS frequently underperforms the load-agnostic RR (*e.g.,* 13.7% vs. 45.8% on Zeta-256). This counter-intuitive result confirms that acting on *stale* information (ILS) creates severe "herding," where multiple requests are routed to the same seemingly idle instance, whereas SIBYL's predictive lookahead effectively resolves this contention.

Table 6. Optimal assignment ratio comparison across different cluster scales.

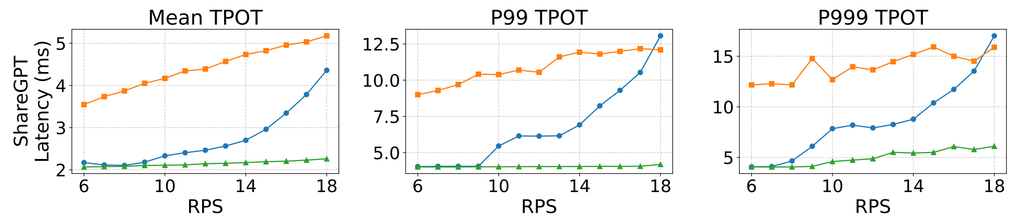
Dataset	# Instances	RR	ILS	SIBYL
ShareGPT	128	60.0%	56.6%	**91.2%** ($\uparrow$1.52×)
	256	71.8%	47.2%	**93.0%** ($\uparrow$1.30×)
Zeta	128	34.3%	16.8%	**49.3%** ($\uparrow$1.44×)
	256	45.8%	13.7%	**59.0%** ($\uparrow$1.29×)



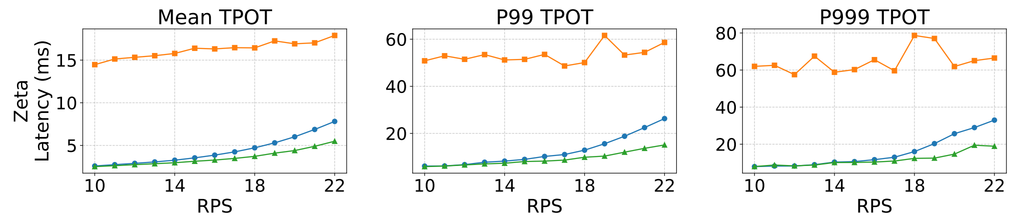
(a) ShareGPT (128 instances)



(b) Zeta (128 instances)



(c) ShareGPT (256 instances)



(d) Zeta (256 instances)

Figure 15. Simulation performance on scaled clusters with 128 and 256 decode instances.

J. Definitions of Sensitivity Variants

In this section, we provide precise definitions for the variants evaluated in the sensitivity analysis (Section 5.3). To isolate the contribution of each modeling component in SIBYL, we modify the objective function or the load estimation logic as follows:

PROB@1 (No Probabilistic Weighting): This variant disables the survival analysis module. Instead of computing the conditional survival probability based on historical distributions, we assume a deterministic worst-case scenario where every active task persists until the completion of the incoming request's prefill phase. Mathematically, we set the probability term $\mathcal{P} = 1$ in Eq. (6) and Eq. (7). This baseline evaluates the necessity of risk-adjusted load estimation versus conservative over-provisioning.

w/o Type 1 (No Active Task Projection): We exclude the projected load from currently running active tasks by setting $\mathcal{L}_{\text{I}}(k, \tau) = 0$. This variant forces the scheduler to ignore the residual workload on decode instances, relying solely on the estimated pressure from pending (shadow) tasks. This tests whether knowing the state of currently executing requests is critical for decision-making.

w/o Type 2 (No Pre-Handoff Shadow): We ignore the ”invisible” contention from requests that are currently in the prefill phase but are projected to start decoding *before* the target request arrives ($\tau_k \leq \tau$). By setting $\mathcal{L}_{\text{II}}(k, \tau) = 0$, the scheduler fails to account for the queue that will form during the prefill interval. This baseline highlights the impact of the temporal gap described in Section 2.2.

w/o Type 3 (No Post-Handoff Shadow): We exclude the interference decay model for tasks projected to start *after* the handoff moment ($\tau_k > \tau$). By setting $\mathcal{L}_{\text{III}}(k, \tau) = 0$, the scheduler optimizes only for the expected load at moment τ, ignoring imminent future arrivals that might cause congestion shortly after.

REQ LEVEL (Request-Level Balancing): This variant abandons the token-based load metric entirely. Instead of modeling physical memory pressure (token occupancy) and duration, it treats all requests as identical units. The scheduler assigns the new request to the instance with the fewest total tasks, regardless of their generation lengths or memory footprint. Notably, unlike the ILS baseline, this variant explicitly accounts for the count of pending requests currently in the prefill phase. This serves as a baseline to validate the necessity of fine-grained, token-aware modeling in heterogeneous LLM workloads.