HyGen: Efficient LLM Serving via Elastic Online-Offline Request Co-location

Ting Sun[*1], **Penghan Wang**[*2], **Fan Lai**[1]
[1] Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign
[2] Department of Computer Science, Purdue University
suntcrick@gmail.com, wang6199@purdue.edu, fanlai@illinois.edu

Abstract

Large language models (LLMs) have facilitated a wide range of applications with distinct service-level objectives (SLOs), from latency-sensitive online tasks like interactive chatbots to throughput-oriented offline workloads like data synthesis. The existing deployment model, which dedicates machines to each workload, simplifies SLO management but often leads to poor resource utilization.

This paper introduces HyGen, an interference-aware LLM serving system that enables efficient co-location of online and offline workloads while preserving SLOs. HyGen incorporates two key innovations: (1) performance control mechanisms, including a latency predictor to estimate batch execution time and an SLO-aware profiler to quantify latency interference, and (2) SLO-aware offline scheduling policies that maximize serving throughput and prevent starvation. Our evaluation on production workloads shows that HyGen achieves up to 3.9-5.8× throughput gains over online and hybrid serving baselines, while ensuring latency SLOs. The code of HyGen is publicly available at https://github.com/UIUC-MLSys/HyGen.

1 Introduction

Large language models (LLMs) have emerged as transformative tools across diverse domains, handling both latency-critical online requests (e.g., chatbot interactions [32, 39]) and throughput-oriented offline tasks (e.g., document summarization [18]). Online serving demands low and stable response times, measured by Time to First Token (TTFT) and Time Between Tokens (TBT), while offline tasks prioritize high throughput and resource utilization, often processing large batches with relaxed latency constraints. The disparity in these requirements has led most production deployments to segregate online and offline serving onto separate clusters to avoid interference [41, 51, 54, 69].

However, real-world LLM workloads demonstrate significant temporal variations in request load. Our analysis of production traces shows that in addition to the popular diurnal request arrival patterns, online request rates can vary by up to 3× within minutes (Section 3). To meet latency requirements under such bursty loads, service providers have to provision GPU resources for peak demand, as minute-level resource scaling is often impractical due to infrastructure complexity and engineering overhead [15, 64, 66]—leading to substantial underutilization during off-peak hours.

This resource overprovisioning suggests an opportunity to improve resource efficiency via hybrid serving—co-locating online and offline workloads on the same inference engine instance. By opportunistically padding online requests with offline requests during periods of low online load, the system could maintain high GPU utilization while preserving latency guarantees for online requests. However, realizing this opportunity requires addressing several fundamental challenges.

[*]Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

First, LLM services exhibit diverse latency requirements across applications [32]. Interactive chatbots require consistent response times with both low initial latency (TTFT) and smooth token generation (TBT), while batch processing tasks prioritize throughput over latency. These requirements often manifest in different statistical metrics—from strict P99 latency bounds to mean performance targets—making it difficult to establish unified resource-sharing policies.

Second, LLM workloads are inherently unpredictable in both their arrival patterns and resource demands. Request arrival rates exhibit both diurnal patterns and unpredictable short-term fluctuations. This variability is further complicated by uncertainty in resource demands—input sequences vary widely in length, and the number of output tokens can hardly be predicted until generation completes.

Third, co-locating online and offline workloads introduces interference. For instance, offline requests may delay time-sensitive online requests; using large batch sizes to improve the throughput of offline requests can increase the latency of online requests. Effectively managing this interference while ensuring latency guarantees demands meticulous orchestration of resource sharing.

This paper presents HyGen, an interference-aware LLM serving system that elastically co-locates online and offline workloads. HyGen introduces several key techniques: (1) a latency predictor that accurately estimates the execution time of different request batches, (2) an interference-aware profiler that quantifies the performance interference of co-location, and (3) an adaptive scheduler that maximizes offline throughput while maintaining strict latency guarantees for online requests.

Our evaluation on production workloads shows that HyGen improves serving throughput by 3.87-5.84$\times$ over existing advances [2], while guaranteeing strict SLO compliance. To summarize, this paper makes the following contributions:

- Key insights on the feasibility and benefits of co-locating online and offline LLM workloads, derived from systematic characterization of production traces.
- A statistical latency prediction model that accurately captures the relationship between batch composition and execution latency, accounting for quadratic complexity in prefill and linear scaling in decode phases.
- A novel scheduling system that dynamically co-locates online and offline workloads, formulated as a constrained optimization problem that maximizes throughput with prefix sharing while preserving strict latency SLOs and providing theoretical guarantees for fairness.
- Experimental evaluation on real-world workloads demonstrating up to 5.84$\times$ throughput improvement over state-of-the-art alternatives.

2 Related Work

LLM Inference Optimization. Recent advances in LLM inference through kernel optimization [12, 61], compilation frameworks [33, 59, 72], and scheduling algorithms [30, 31, 55] have substantially improved LLM serving performance. Among these, a particularly important development is predictive scheduling, which aims to estimate request difficulty [13, 38] or generation length [16, 23, 27, 43, 45]. Contrasting these efforts, HyGen uniquely predicts batch execution time, providing precise control over request interference during co-located workloads. Additionally, the concept of request prefix sharing has been explored by [17, 28, 71], where shared prefixes across requests optimize resource use. HyGen takes this a step further by applying a prefix sharing maximization strategy to opportunistically schedule offline requests, utilizing residual capacity from a primary, SLO-bound online workload. Moreover, HyGen introduces a fairness-aware extension to the standard prefix-sharing maximization (PSM) method, addressing the issue of starvation often encountered in naive designs.

In parallel, the increasing demand for offline LLM inference has driven two key trends. First, platforms like Huggingface Accelerate [25], DeepSpeed ZeRO-Inference [4], and FlexGen [51] enable efficient inference on commodity hardware through memory offloading across GPUs, CPUs, and disk. Second, cloud providers, including OpenAI's Batch API [40], have launched specialized services optimized for throughput rather than latency, processing millions of requests daily for large-scale data processing tasks.

Workloads Co-location. In data center environments, co-locating latency-sensitive applications with batch applications has been explored to improve resource utilization [8, 9, 42, 68]. However,

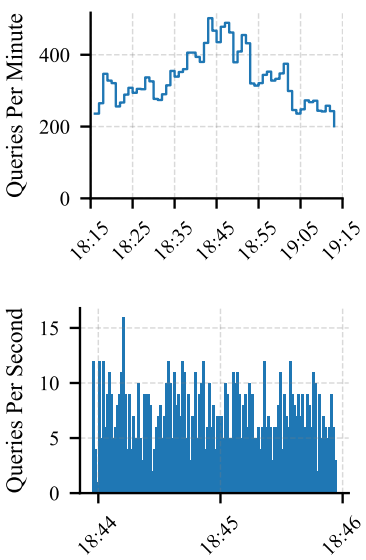


Figure 1: Request rate varies significantly in Microsoft Azure's LLM service over one-hour and two-minute periods.

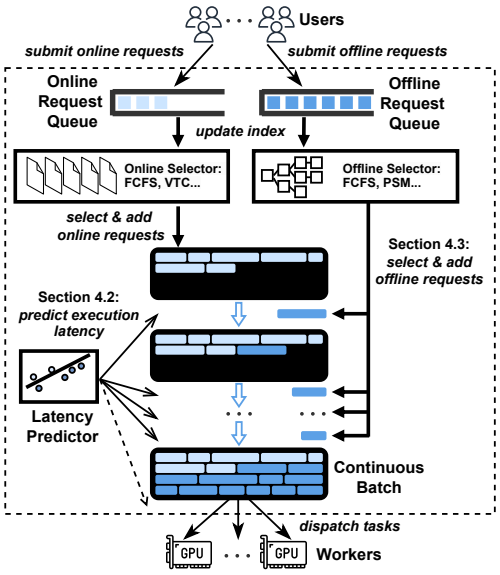


Figure 2: HyGen Overview. Online and offline requests are processed asynchronously, with offline requests opportunistically scheduled to respect latency budgets.

these approaches generally overlook the unique characteristics of LLM inference. In the context of LLM inference, several works have investigated the co-location of various model types and tasks. For instance, Punica [7], S-LORA [49], and dLoRA [58] batch requests from different LoRAs, while MuxServe [14] multiplexes resources across multiple LLMs. In contrast, HyGen focuses on batching online and offline requests, addressing distinct optimization problems that are not interchangeable.

3 Background and Motivation

This section first introduces the background of LLM serving deployment (Section 3.1), then illustrates how these characteristics motivate our system designs (Section 3.2).

3.1 LLM Serving

Large-scale inference clusters consist of multiple serving instances, with a router intelligently directing incoming requests to the most suitable instances [53, 65]. Each instance, which could be based on architectures like vLLM [29] or SGLang [71], typically employs iteration-level scheduling [63] and chunked prefill [2]. This setup enables decode requests to perform an additional decoding step, while prefill requests are limited to a fixed token budget. These two types of requests are processed together within a single iteration, optimizing resource utilization.

LLM serving deployment can be categorized into online serving and offline serving scenarios. Online serving targets real-time user interactions, such as chatbots, code assistants, and interactive applications [35, 39]. This interactive nature often requires a short Time to First Token (TTFT) as well as a short Time Between Tokens (TBT). Offline serving prioritizes throughput over latency, measured in queries per second (QPS) or tokens per second (TPS). For example, OpenAI's Batch API processes requests with relaxed latency requirements (up to 24 hours) at significantly lower costs compared to standard APIs [40]. Applications include model benchmarking [21, 34], document processing [10, 67], data cleaning [36, 70], and data synthesis [3].

3.2 Motivation and Challenges

LLM serving systems face a critical resource utilization challenge due to the highly variable nature of their workloads. Figure 1 reports our analysis of Microsoft Azure's LLM service [41], showing that

request rates can fluctuate dramatically—varying up to 3× within minutes while following broader diurnal patterns. This variability creates an inherent tension in resource provisioning: serving clusters must be sized to handle peak loads, leading to resource underutilization during off-peak periods.

This load variability suggests an opportunity to improve resource utilization by co-locating online requests with offline requests. For each serving instance in the cluster, during periods of low online traffic, it can opportunistically schedule offline tasks to harvest idle resources. While recent methodologies in simultaneous batching of prefill and decode requests set the premise for dynamic request co-location [2], doing so at scale introduces several fundamental challenges:

1. *Diverse Latency Requirements:* Applications and even requests of an application have distinct latency requirements. For example, paid users require strict latency SLOs while free users accept more relaxed guarantees. How to respect diverse latency requirements in flight?
2. *Massive Uncertainties:* LLM serving faces temporal uncertainty—online requests arrive in unpredictable bursts with varying urgency levels—as well as resource demand uncertainty due to unpredictable output lengths. How to perform efficient scheduling in the wild?
3. *Request Interference:* Co-locating online and offline workloads introduces performance interference. Large batches of offline requests can cause severe head-of-line blocking, delaying the processing of time-sensitive online requests. Worse, batching requests of long inputs with short interactive queries can elongate the latency of all requests in the batch by an order of magnitude [2]. How to account for interference in co-locating requests?

4 The HyGen Design

4.1 Overview

HyGen introduces a novel approach to integrate online and offline requests while maintaining strict latency guarantees. As shown in Figure 2, HyGen employs a dual-queue architecture that separates latency-sensitive and throughput-oriented requests. This design accommodates diverse SLOs and variable workloads while remaining compatible with existing scheduling policies within each queue. We note that HyGen functions as an instance-level scheduler, receiving requests from an upstream system-level router (e.g., Preble [53]). As a result, both the request concurrency and scheduling overhead at each instance are inherently bounded. HyGen's two-phase scheduling operates as follows (see Appendix A.1 for the asynchronous two-queue workflow and message passing details):

- The online phase prioritizes latency-sensitive requests, forming an initial batch using established policies such as First-Come-First-Serve (FCFS) [29] or fairness request scheduling [50]. We introduce a priority-based preemption mechanism that protects online request performance by selectively preempting offline requests. Currently, HyGen preserves execution state for preempted requests, while its architecture supports various preemption mechanisms—including state discarding, preservation, and swapping—as categorized by InferCept [1].
- The offline phase uses our latency predictor to allocate remaining capacity to throughput-oriented requests. This predictor accurately estimates the latency impact of each potential offline request addition, determining either decode request latency costs or maximum chunked prefill lengths that fit within the available latency budget without violating online SLOs.

Algorithm 1 formalizes our scheduling approach, with implementation details and complexity analysis ($O(n)$ where n is the number of requests) provided in Appendices A.1 and A.4. Appendix C presents a novel cluster serving paradigm that addresses the longstanding tradeoff between SLO compliance and resource utilization based on HyGen.

4.2 Performance Control Mechanisms

Co-locating online and offline requests under diverse SLO requirements requires precise control over resource allocation. The key challenge lies in accurately estimating the latency impact of scheduling decisions to ensure SLO compliance. This section introduces a latency predictor for estimating batch execution latency and an SLO-aware profiler for translating the estimates into scheduling decisions.

Algorithm 1 HyGen SLO-aware scheduler

function SLO_AWARE_SCHEDULE
  Input: running requests R, request queue Q,
    latency budget t, chunk size c, memory budget m
  Output: batched requests B
  $B \leftarrow \{\}$
  for $r \in R$.decode **do**
    $t_{req} \leftarrow$ PREDICTOR.predict(r, DECODE) // predict latency of the decoding request
    if $t_{req} \leq t$ or PHASE == ONLINE **then**
      // schedule request if it is: 1. online, or 2. offline and enough latency budget left
      $t \leftarrow t - t_{req}$
      $B \leftarrow B \cup \{(r, 0, t_{req})\}$
  for $r \in R$.prefill $\cup Q$ **do**
    TRY_SCHEDULE: // try to schedule a prefilling or waiting request
    // get the max number of tokens allowed under memory and latency budget
    $l, t_{req} \leftarrow$ PREDICTOR.get_max_tokens(t, c, m, r)
    if $l > 0$ **then**
      // schedule request
      $t \leftarrow t - t_{req}$
      $c \leftarrow c - l$
      $m \leftarrow m -$ GET_NUM_BLOCKS(l)
      $B \leftarrow B \cup \{(r, l, t_{req})\}$
    else
      if PHASE == ONLINE and $R \neq B$ **then**
        PERFORM_PREEMPTION(R, m) // preempt request with lower priority
        goto *TRY_SCHEDULE* // try to schedule again
      else
        break
  return B

Latency Predictor. The design of our latency predictor is guided by three key requirements. First, it must provide fast inference to support real-time scheduling decisions. Second, it needs to be robust across varying workload patterns to maintain reliable performance. Third, it should be adaptable to different hardware configurations to accurately capture their unique performance characteristics.

The execution time of an LLM serving batch is primarily determined by two distinct processing stages with different computational patterns. The prefill stage exhibits quadratic complexity due to attention computations, with latency growing quadratically with input sequence length. The total load of this stage depends on both the number of requests and their individual sequence lengths. In contrast, the decode stage shows linear scaling, with computational requirements growing proportionally with the number of tokens. We can model the batch execution time below to capture these characteristics:

$$T_{batch} = f(S_p, S_d, S_p^2, S_d^2, N_p, N_d) \tag{1}$$

where S_p and S_d represent the total number of prefill and decode tokens in the batch, respectively. The quadratic terms (S_p^2 and S_d^2) account for non-linear scaling effects, particularly in the prefill phase. N_p and N_d represent the number of prefill and decode requests in the batch, respectively.

We employ linear regression as the prediction model because of its efficiency and effectiveness. Training data for the model is collected by systematically profiling target hardware across diverse batch compositions, varying in the number of requests in different phases, sequence length distributions, and total batch sizes. The linear model enables rapid evaluation of varying batch compositions during scheduling, while its simple feature set ensures stable predictions across varying conditions. Further discussion on the implementation and expandability of the LR predictor can be found in Appendix B.

SLO-aware Profiling. The latency predictor provides accurate latency estimates for filling offline requests. Our SLO-aware profiler leverages a latency budget to ensure SLO compliance in scheduling. The profiler first analyzes the given combination of workload and SLO to establish viable latency

budget ranges. Given that larger batch sizes and longer inputs will increase latency, the profiler test-runs latency budgets within the range to check their compliance with the given SLO and employs binary search to decide an upper limit that meets the overall SLO for online requests. During deployment, this latency budget is used as the batch latency limit in the two-phase scheduling process (Section 4.1) to ensure SLO compliance. This profiling enables three key capabilities: (1) It determines appropriate latency thresholds that maintain SLO compliance for various workloads and limitations (e.g., power constraints [41]). (2) It provides flexible adaptation by adjusting budgets based on changing workload characteristics and performance requirements. (3) It establishes a robust foundation for hybrid scheduling by accounting for both online and offline workload patterns.

4.3 SLO-aware Offline Scheduling Policies

After scheduling online requests, our offline scheduling policy repurposes the residual capacity to maximize throughput while ensuring fairness [50]. To further optimize the serving throughput for offline requests, HyGen employs an SLO-aware Prefix Sharing Maximization (PSM) strategy. Prefix sharing is a widely adopted technique for reusing the KV cache of shared input prefixes between requests [17, 69, 71, 73].

Prefix Sharing Maximization Strategy. Our PSM strategy organizes offline requests into a prefix tree following the structure of a Trie tree with each leaf node representing a request, capturing prefix sharing characteristics of all offline requests. The priority of each request is determined by the Depth-First Search (DFS) order of the prefix tree, where requests with the greatest prefix sharing potential are scheduled together. Subsequently, HyGen performs SLO-aware offline scheduling (Section 4.1) using this order to maximize prefix cache reuse, reducing redundant computation and improving throughput. The prefix tree structure also ensures fast insertion and deletion in runtime scheduling. A formalized algorithm is in Appendix A.2. For example, consider a system that can process two offline requests per batch with the following request queue: (*What is ML*, *How to code*, *What is AI*, *How to debug*). Under traditional FCFS scheduling, requests are processed in arrival order: (*What is ML*, *How to code*), (*What is AI*, *How to debug*), resulting in no prefix sharing opportunities. In contrast, PSM's prefix-aware scheduling reorders requests as: (*What is ML*, *What is AI*), (*How to code*, *How to debug*), enabling KV cache reuse through shared prefixes.

The PSM strategy demonstrates strong extendability. Under certain scenarios, the vanilla PSM strategy may lead to starvation for requests with minimal prefix-sharing potential. Consider a request queue: (*What is ML*, *What is AI*, *How to code*, *What is DL*). When new requests arrive with similar prefixes (*What is LLM*, *What is DNN*), a naive prefix-sharing policy would continuously prioritize requests sharing the *What is* prefix, potentially starving the *How to code* request indefinitely. This issue can be mitigated by an extended version of our PSM policy, combining maximum prefix sharing with request freshness by using a utility ratio to ensure a balance between efficiency and fairness. Based on the utility ratio, a new offline request would be selected from the DFS order of the prefix tree or the most stale request from a self-balanced binary search tree sorted by freshness. A detailed algorithm is in Appendix A.3. These enhancements can make the PSM strategy more practical for real-world deployments, retaining its efficiency while improving fairness and adaptability.

5 Performance Evaluation

5.1 Evaluation Setup

Implementation and Testbeds. We implement HyGen on top of vLLM [29, 57] and Sarathi [2, 46], with 1,300 lines of additional code. We evaluate HyGen on three server configurations: one with 4 NVIDIA A100 GPUs (40GB VRAM each), one with 4 NVIDIA A40 GPUs (48GB VRAM each), and one with 1 NVIDIA A5000 GPU (24GB VRAM). All servers have 64 CPU cores, 256GB DDR4 RAM, and a 1.5TB NVMe SSD.

Models and Workloads. For end-to-end evaluation, we use Llama2-7B [56] and Qwen-14B [6] models on A100 and A40 GPUs, respectively. Online workloads are based on the conversation trace from Azure LLM inference trace 2023 [41], a one-hour production trace with real-world requests and timestamps. We randomly sampled the trace to achieve the desired QPS that suits our hardware serving capacity. Specifically, within a time duration of T seconds, we would sample $T \times Q$ requests

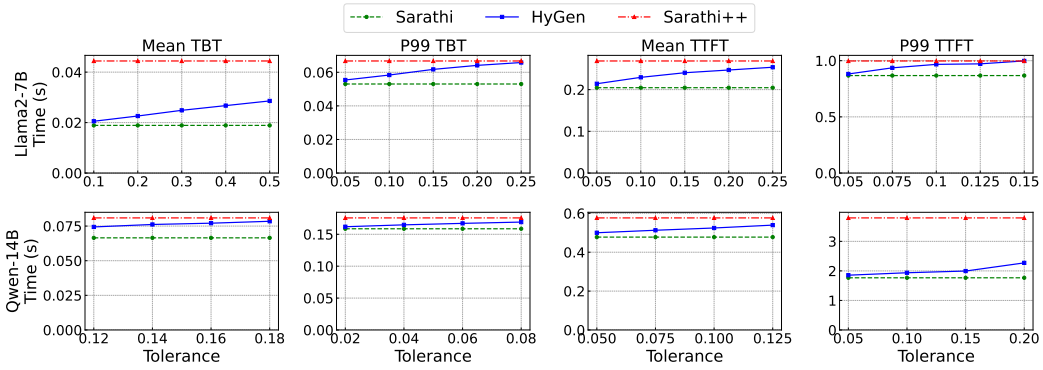


Figure 3: HyGen respects latency requirements in co-locating requests.

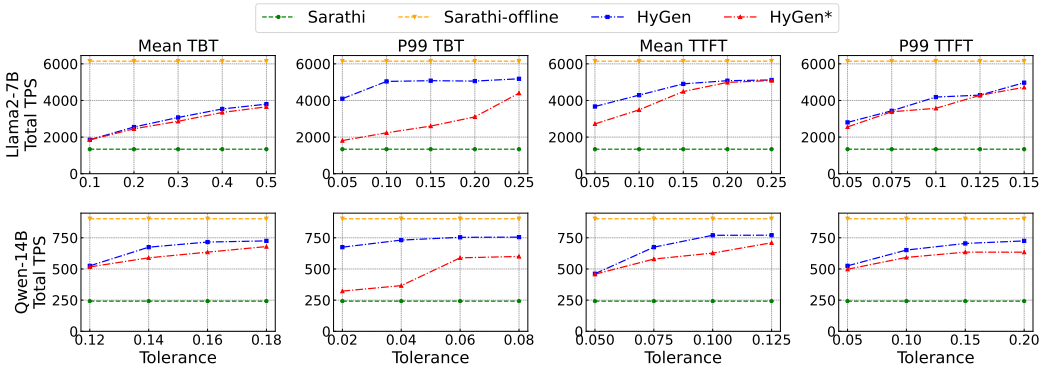


Figure 4: HyGen improves serving throughput under varying SLOs.

to suit a desired QPS Q. For offline workloads, we use arXiv summarization [11], a dataset for long document summarization. In our ablation studies, we evaluate HyGen across different model scales ranging from Sheared-LLaMA-2.7B [60], Mistral-7B [26] to Yi-34B [62]. The Mooncake trace [44] is further used as the online trace, providing industrial request length distributions and arrival patterns, while CNN/DailyMail [22, 47] and MMLU [20, 21] are used as offline traces in the ablation studies. For interference evaluation, we focus on Time to First Token (TTFT) and Time Between Tokens (TBT), including their mean and 99th percentile (P99) values. Throughput is measured in tokens per second (TPS) and queries per second (QPS).

Baselines. For pure online inference, we use Sarathi [2] as our baseline. For pure offline serving, we use *Sarathi-offline* to evaluate the maximum offline serving capacity, where an optimal chunk size is profiled for offline workload to maximize throughput. The hyperparameter search of *Sarathi-offline* achieves $\sim$12% throughput gain compared to the default setup, ensuring optimal baseline performance for fair comparison. We then compare HyGen with two baselines for interference and throughput evaluation, respectively: (1) *Sarathi++*: We implement our online-first scheduling policy on Sarathi to support hybrid serving, including the request management and preemption handling policies introduced in Section 4.1. (2) *HyGen**: To evaluate the throughput benefit of our HyGen design, we further improve *Sarathi++* to an SLO-aware serving system, *HyGen**. Besides inheriting the serving policies from *Sarathi++*, *HyGen** serves offline requests at a specific offline QPS to control overall interference. The offline QPS is profiled using a similar design with the HyGen profiler to guarantee bounded SLO interference.

5.2 End-to-end Performance

HyGen respects latency requirements in co-locating requests. We evaluate HyGen under four SLO metrics (mean TBT, P99 TBT, mean TTFT, and P99 TTFT) with varied interference tolerance ratios. Figure 3 shows that HyGen controls interference and guarantees to meet specific SLOs across our settings. Compared with *Sarathi++*, an SLO-unaware system that yields the same result for all metrics and tolerance ratios, HyGen shows efficient SLO-aware latency control.

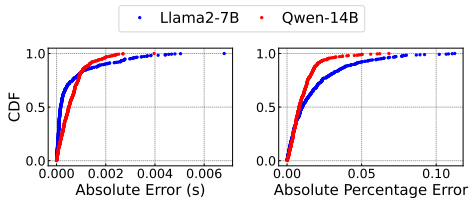


Figure 5: HyGen latency predictor achieves high accuracy for batch latency prediction.

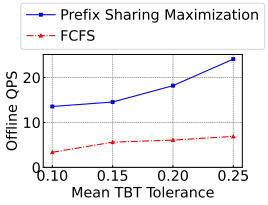


Figure 6: Prefix Sharing Maximization improves serving throughput.

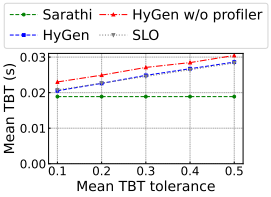


Figure 7: SLO-aware profiler contributes to HyGen's improvements.

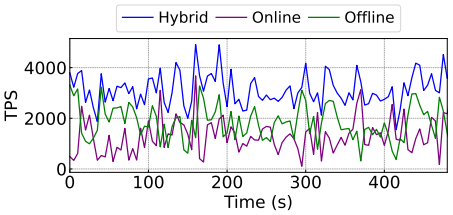
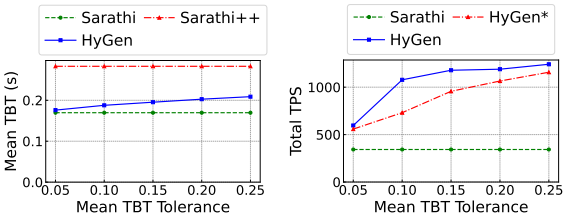


Figure 8: HyGen dynamically controls throughput according to online workload.



(a) SLOs (b) Throughput

Figure 9: HyGen meets SLOs and achieves higher throughput for Yi-34B model using TP=2, PP=2.

HyGen improves serving throughput. Figure 4 shows the offline throughput of HyGen for various metrics and tolerance ratios. Through efficient request co-location, HyGen improves overall serving throughput by up to 3.87× compared to pure online serving. Under the same SLO, HyGen consistently achieves higher throughput compared to *HyGen**, yielding up to $5.84\times$ offline throughput gain. Furthermore, HyGen achieves up to 84.3% total throughput compared to *Sarathi-offline*, a pure offline serving system whose high throughput benefits from an optimal chunk size profiled for offline requests only. This verifies that our fine-grained latency predictor and SLO-aware profiler designs achieve higher serving efficiency compared to their simplified counterparts.

5.3 Performance Breakdown

Accuracy of latency predictor. We evaluate the accuracy of our latency predictor on Llama2-7B and Qwen-14B using Azure LLM Inference trace mixed with arXiv summarization dataset. Figure 5 shows that our latency predictor achieves a mean absolute percentage error of only 1.78% and 1.07%, confirming its high accuracy.

Impact of prefix sharing. To test HyGen's compatibility with prefix sharing, we conducted a simulation experiment using Azure LLM Inference as the online trace and MMLU [20, 21] as the offline dataset on a Llama2-7B model. In our simulation, we deducted the shared prompt prefix length between consecutive offline requests to simulate prefix sharing. Figure 6 shows that HyGen yields up to $4\times$ offline throughput gain with its prefix sharing maximization scheduling policy.

Impact of SLO-aware profiler. To demonstrate the effect of HyGen's SLO-aware profiler, we compared it with a simple strategy that sets the desired mean TBT SLO as the batch latency budget. Figure 7 shows the performance gap between individual batch latencies and overall mean TBT, illustrating how the SLO-aware profiler bridges this gap for controlled SLO in hybrid serving.

Breakdown by time. Figure 8 shows a temporal throughput breakdown of HyGen. At runtime, HyGen dynamically adjusts offline throughput based on online workload and overall latency budget, batching offline requests more aggressively during online QPS troughs and reducing offline throughput during online bursts, harnessing compute resources in an adaptive manner.

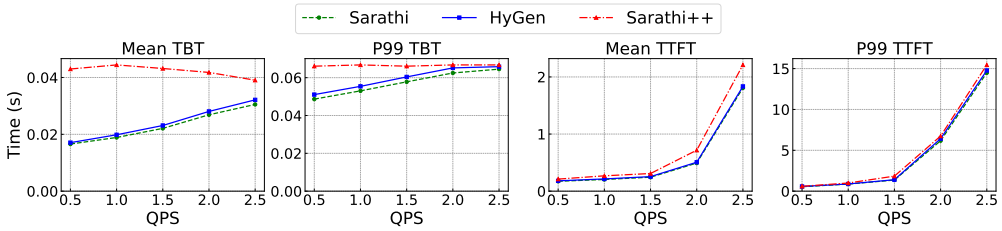


Figure 10: HyGen meets SLO under various online QPS settings.

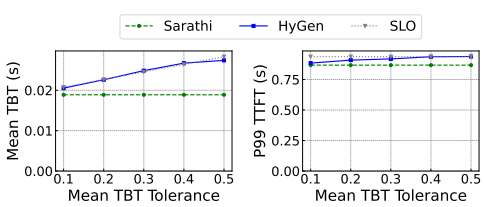
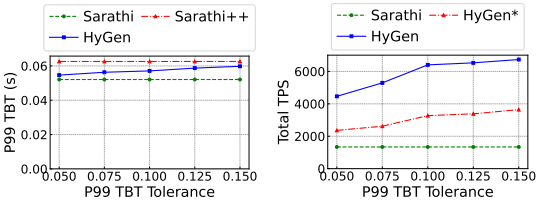


Figure 11: HyGen is able to meet multiple SLOs simultaneously.



(a) SLOs (b) Throughput

Figure 12: HyGen meets SLOs and achieves higher throughput with CNN/DailyMail offline dataset.

5.4 Ablation Studies

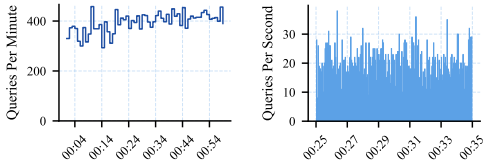
Impact of model parallelisms. To evaluate HyGen's effectiveness in distributed inference, we deployed the Yi-34B model [62] on a server with 4 NVIDIA A40 GPUs using tensor-parallelism (TP) [52] and pipeline-parallelism (PP) [5, 24, 37] with degree 2 for each dimension. Using Azure LLM Inference and arXiv summarization workloads, Figure 9 shows that HyGen maintains its ability to meet SLOs and achieves higher offline throughput (up to $1.89\times$) than the baseline.

Impact of SLO requirements. We further evaluate HyGen's ability to meet stringent SLOs under varying online QPS settings on the four aforementioned metrics, each with 5% interference tolerance. Figure 10 shows that HyGen meets stringent SLOs for all metrics. We further demonstrate HyGen's ability to meet multiple SLOs at the same time. By testing HyGen with a fixed P99 TTFT interference ratio (8%) and mean TBT interference ratios ranging from 10% to 50%, Figure 11 shows that at a lower mean TBT tolerance, HyGen's performance is bounded by mean TBT SLOs; after reaching the fixed P99 TTFT SLO, mean TBT stops increasing in order to keep P99 TTFT under control.

Impact of models and datasets. We further evaluate HyGen's adaptability on two more experiments: The first using Mistral-7B model [26] with Mooncake trace [44], a trace containing request lengths and timestamps taken from real-world servers, as the online trace, and arXiv summarization as the offline trace; The second experiment uses Llama2-7B model with Azure LLM Inference trace and replaced the offline dataset with CNN/DailyMail summarization dataset [22, 47]. Figure 13 shows the varying request arrival rates of Mooncake trace over one-hour and ten-minute periods, further demonstrating the fluctuating nature of LLM services. Figure 12 and Figure 14 show that HyGen achieves superior performance than its counterparts under these settings.

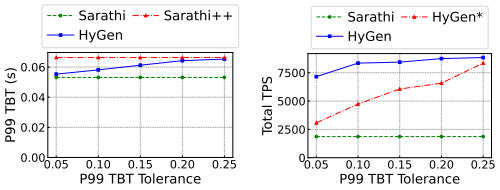
Impact of hardware testbeds. To further evaluate HyGen's effectiveness under different hardware configurations, memory limitations and model sizes, we further conducted experiments on A5000 GPU with 24 GB VRAM and Sheared-LLaMA-2.7B model [60]. Figure 15 shows that HyGen is able to guarantee SLO attainment and achieve higher throughput, with up to $2.18\times$ offline throughput gain and $1.30\times$ overall throughput gain compared to the baseline.

Impact of predictor accuracy. We tested HyGen's robustness using several pre-trained LR latency predictors with varying prediction accuracy taken from other workloads and tested them on Azure LLM Inference trace and arXiv summarization dataset. Figure 16 shows how predictor accuracy (measured in mean absolute percentage error) affects offline throughput under the same P99 TBT



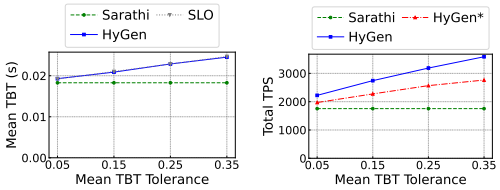
(a) 1 hour (b) 10 minutes

Figure 13: Request rate varies in Moonshot Mooncake's LLM service over one-hour and ten-minute periods.



(a) SLOs (b) Throughput

Figure 14: HyGen meets SLOs and achieves higher throughput for Mooncake trace.



(a) SLOs (b) Throughput

Figure 15: HyGen meets SLOs and achieves higher throughput on A5000 GPU and Sheared-LLaMA-2.7B model.

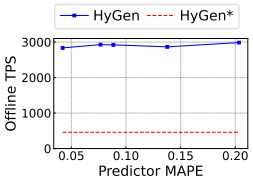


Figure 16: HyGen is robust to predictor accuracy.

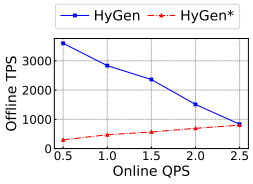


Figure 17: HyGen dynamically adjusts offline throughput.

SLO. HyGen remains robust across different accuracy settings. Also, our LR-based latency predictor is lightweight for training, with only $\sim$15ms training time for over 80,000 samples on CPUs. HyGen's lightweight latency predictor also only incurs $\sim$18μs runtime latency per iteration on our experiment CPU, guaranteeing efficient runtime scheduling.

Impact of online arrival rate. Figure 17 shows the effect of online QPS on offline throughput with 5% P99 TBT tolerance. As online load increases, HyGen adjusts offline throughput based on the system's residual serving capacity while maintaining higher throughputs. Understandably, a high online arrival rate limits the headroom for co-location as it approaches system serving capacity.

6 Conclusion

This paper introduces HyGen, an LLM serving system that enables efficient co-location of online and offline workloads. We employ control mechanisms to predict and manage interference impacts, and a scheduling policy to opportunistically schedule offline serving. Evaluation on production workloads demonstrates that HyGen improves serving throughput by 3.87-5.84$\times$ while maintaining strict latency SLOs.

Acknowledgements

We thank the anonymous reviewers for their constructive and insightful feedback. This work was supported in part by grants from Cisco and Google, and by an award from NVIDIA Academic Program. It also utilized the Delta system at the National Center for Supercomputing Applications (NCSA) through allocation CIS240236 from the ACCESS program.

References

[1] Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. Infercept: Efficient intercept support for augmented large language model inference. *arXiv preprint arXiv:2402.01869*, 2024.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff

in llm inference with sarathi-serve. In *Proceedings of 18th USENIX Symposium on Operating Systems Design and Implementation, 2024, Santa Clara*, 2024.

[3] Loubna Ben Allal, Anton Lozhkov, and Daniel van Strien. Cosmopedia: how to create large-scale synthetic data for pre-training Large Language Models — huggingface.co. `https://huggingface.co/blog/cosmopedia`, 2024. [Accessed 25-10-2024].

[4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2022.

[5] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 472–487, 2022.

[6] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[7] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. *Proceedings of Machine Learning and Systems*, 6:1–13, 2024.

[8] Ruobing Chen, Haosen Shi, Yusen Li, Xiaoguang Liu, and Gang Wang. Olpart: Online learning based resource partitioning for colocating multiple latency-critical jobs on commodity computers. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 347–364, 2023.

[9] Shuang Chen, Christina Delimitrou, and José F Martínez. Parties: Qos-aware resource partitioning for multiple interactive services. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 107–120, 2019.

[10] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. Spreadsheetcoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*, pages 1661–1672. PMLR, 2021.

[11] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[13] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618*, 2024.

[14] Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. Muxserve: Flexible spatial-temporal multiplexing for multiple llm serving. In *Forty-first International Conference on Machine Learning*, 2024.

[15] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. {ServerlessLLM}:{Low-Latency} serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 135–153, 2024.

[16] Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient llm scheduling by learning to rank. *arXiv preprint arXiv:2408.15792*, 2024.

[17] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

[18] Grand View Research. Large language model (llm) market size, share & trends analysis report by component, by application, by enterprise size, by end-use, by region, and segment forecasts, 2023 - 2030. Grand View Research, 2023.

[19] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[20] Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[22] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28, 2015.

[23] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, et al. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024.

[24] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[25] Hugging Face. Hugging face accelerate. GitHub repository, 2025. Accessed: 2025-01-01.

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

[27] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

[28] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y Fu, Christopher Ré, and Azalia Mirhoseini. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

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

[30] Bin Lin, Tao Peng, Chen Zhang, Minmin Sun, Lanbo Li, Hanyu Zhao, Wencong Xiao, Qi Xu, Xiafei Qiu, Shen Li, et al. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*, 2024.

[31] Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, 2024.

[32] Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*, 2024.

[33] Zixuan Ma, Haojie Wang, Jingze Xing, Shuhong Huang, Liyan Zheng, Chen Zhang, Huanqi Cao, Kezhao Huang, Mingshu Zhai, Shizhi Tang, et al. Intelligen: Instruction-level auto-tuning for tensor program with monotonic memory optimization. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization*, pages 107–122, 2025.

[34] Meta-Team. The llama 3 herd of models, 2024.

[35] Microsoft. GitHub Copilot · Your AI pair programmer — github.com. `https://github.com/features/copilot`, 2023. [Accessed 28-10-2024].

[36] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can foundation models wrangle your data? *Proc. VLDB Endow.*, 16(4):738–746, December 2022.

[37] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[38] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*, 2024.

[39] OpenAI. Introducing chatgpt. `https://openai.com/index/chatgpt/`, 2022. [Accessed 20-10-2024].

[40] OpenAI. Batch api, 2024.

[41] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[42] Tirthak Patel and Devesh Tiwari. Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 193–206. IEEE, 2020.

[43] Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. Queue management for slo-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*, pages 18–35, 2024.

[44] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Kimi's kvcache-centric architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.

[45] Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K Iyer. Power-aware deep learning model serving with $\{\mu$-Serve$\}$. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 75–93, 2024.

[46] Sarathi-Serve Project. Sarathi-serve: A low-latency and high-throughput serving engine for llms. GitHub repository, 2024. Accessed: 2025-01-01.

[47] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, 2017.

[48] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

[49] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. Slora: Scalable serving of thousands of lora adapters. *Proceedings of Machine Learning and Systems*, 6:296–311, 2024.

[50] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 965–988, 2024.

[51] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR, 2023.

[52] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[53] Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. *arXiv preprint arXiv:2407.00023*, 2024.

[54] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. Dynamollm: Designing llm inference clusters for performance and energy efficiency. *arXiv preprint arXiv:2408.00741*, 2024.

[55] Ting Sun, Penghan Wang, and Fan Lai. DiSCo: Device-server collaborative LLM-based text streaming services. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 14259–14277, Vienna, Austria, July 2025. Association for Computational Linguistics.

[56] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[57] vLLM Project. vllm: Easy, fast, and cheap llm serving with pagedattention. GitHub repository, 2023. Accessed: 2025-01-01.

[58] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. {dLoRA}: Dynamically orchestrating requests and adapters for {LoRA}{LLM} serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 911–927, 2024.

[59] Mengdi Wu, Xinhao Cheng, Shengyu Liu, Chunan Shi, Jianan Ji, Man Kit Ao, Praveen Velliengiri, Xupeng Miao, Oded Padon, and Zhihao Jia. Mirage: A {Multi-Level} superoptimizer for tensor programs. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 21–38, 2025.

[60] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

[61] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

[62] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

[63] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[64] Minchen Yu, Rui Yang, Chaobo Jia, Zhaoyuan Su, Sheng Yao, Tingfeng Lan, Yuchen Yang, Yue Cheng, Wei Wang, Ao Wang, et al. {\lambda} scale: Enabling fast scaling for serverless large language model inference. *arXiv preprint arXiv:2502.09922*, 2025.

[65] Yifan Yu, Yu Gan, Nikhil Sarda, Lillian Tsai, Jiaming Shen, Yanqi Zhou, Arvind Krishnamurthy, Fan Lai, Hank Levy, and David Culler. Ic-cache: Efficient large language model serving via in-context caching. In *SOSP*. ACM, 2025.

[66] Shaoxun Zeng, Minhui Xie, Shiwei Gao, Youmin Chen, and Youyou Lu. Medusa: Accelerating serverless llm inference with materialization. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 653–668, 2025.

[67] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. SummIt: Iterative text summarization via ChatGPT. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10644–10657, Singapore, December 2023. Association for Computational Linguistics.

[68] Wei Zhang, Zhiyu Wu, Yi Mu, Banruo Liu, Myungjin Lee, and Fan Lai. Tempo: Application-aware llm serving with mixed slo requirements. *arXiv preprint arXiv:2504.20068*, 2025.

[69] Yilong Zhao, Shuo Yang, Kan Zhu, Lianmin Zheng, Baris Kasikci, Yang Zhou, Jiarong Xing, and Ion Stoica. Blendserve: Optimizing offline inference for auto-regressive large models with resource-aware batching. *arXiv preprint arXiv:2411.16102*, 2024.

[70] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

[71] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2024.

[72] Liyan Zheng, Haojie Wang, Jidong Zhai, Muyan Hu, Zixuan Ma, Tuowei Wang, Shuhong Huang, Xupeng Miao, Shizhi Tang, Kezhao Huang, et al. {EINNET}: Optimizing tensor programs with {Derivation-Based} transformations. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 739–755, 2023.

[73] Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024.

Limitations

While HyGen demonstrates substantial improvements in throughput and latency compliance through co-locating online and offline LLM workloads, several limitations remain. First, our approach assumes stable performance predictions from the latency predictor, which may degrade under highly dynamic or adversarial inputs. Second, HyGen focuses on a single model co-location scenario; extending support to heterogeneous models or multi-tenant environments could introduce additional interference patterns. Lastly, our evaluation is limited to specific production workloads—generalizing to other LLM architectures or serving frameworks may require further adaptation and tuning.

Broader Impact

This paper proposes HyGen, a LLM serving system for efficient co-location of online and offline requests. Through efficient co-location and SLO control mechanisms, HyGen improves resource utilization and system serving throughput. We believe that the deployment of HyGen will help any kind of LLM service providers by improving serving throughput and providing LLM service with a wider range of options (online/offline). Since our paper provides an efficient serving system for LLM applications without modification to the structure or the outputs of the model, there are no possible negative societal impact that needs to be mentioned in our paper as far as we are concerned.

Code and Dataset Licenses

Codebase. HyGen's implementation is based on vLLM[57] and Sarathi-Serve[46], both using Apache-2.0 License.

Datasets. We list the license of used datasets as follows:

arXiv summarization dataset[11]: Apache-2.0 License;

Azure LLM Inference trace[56]: CC-BY-4.0;

MMLU dataset [20, 21]: MIT License.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and introduction claim the key innovations and results of our proposed system, HyGen. We have included detailed explanation of our design in Section 4 and presented comprehensive experimental results in Section 5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have provided detailed discussion of the limitations in this paper. We have also conducted holistic ablation studies in Section 5.4 to reflect on each factor of our evaluation setup, including hardware configurations, models, datasets, traces, etc.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our paper does not include major theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We have provided detailed explanation of our evaluation setup for each experiment in Section 5, including testbeds, models, datasets, workloads, etc.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code of HyGen is publicly available at `https://github.com/UIUC-MLSys/HyGen`.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We have included the evaluation setup and details for each experiment in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our main metrics include SLO compliance and throughput gain, which have both been well-defined in previous literature and related works. We have included detailed metrics and data for our experimental results in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have included the testbeds and hardware configurations in Section 5.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirm that our research complies with the Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have provided relevant discussion in the "Broader Impact" section.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our paper presents an efficient serving system for request co-location in LLM serving, which does not involve further training or modification to the model. Our paper does not involve scraped datasets.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have provided explicit information and citation of each asset used in this paper.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The codebase of HyGen is publicly available at `https://github.com/UIUC-MLSys/HyGen` with detailed documentation.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendix

A Algorithms

A.1 HyGen Two-phase Scheduling Algorithm

This section gives a detailed and formalized demonstration of the two-phase scheduling algorithm introduced in Section 4.1. In each scheduling step, the workflow invokes the SLO-aware scheduling process in Algorithm 1 twice (line 13 and line 18) to form a hybrid batch with online and offline request co-location while respecting latency and memory limits. To reduce the scheduling overhead of HyGen, we employ a message queue for asynchronous communication between the main process and the offline scheduler. After each scheduling step, the main process sends the metadata of batched requests to the message queue. The offline scheduler first calculates the expected status of each request based on scheduling decisions from the previous batch, and then runs a scheduling simulation to generate offline request scheduling decisions using our latency predictor and the profiled latency budget. The offline scheduling decisions are then sent back to the main process using the message queue and used for the next scheduling step. To support pipeline parallelization, a scheduling history archive of K steps is kept by the offline scheduler for pipeline parallelization degree K, in order to have a holistic view of every request running in each pipeline stage at the time.

Algorithm 2 HyGen two-phase scheduler

1: **global** Q_{on} (online request queue), Q_{off} (offline request queue)
2: **global** R_{on} (online request running list), R_{off} (offline request running list)
3: **global** Q_{send}, Q_{recv} (message queues)
4: **global** L (latency budget), M (memory budget), C (chunk size), M_{off} (offline memory)
5: **function** ASYNC_SCHEDULER
6: **while** True **do**
7: scheduled requests $S \leftarrow Q_{send}$.get(block=True)
8: UPDATE_REQUEST_STATUS(S, R_{on}, R_{off})
9: batched requests $B \leftarrow \{\}$
10: latency budget $t \leftarrow L$
11: memory budget $m \leftarrow$ GET_FREE_MEMORY() + M_{off}
12: chunk size $c \leftarrow C$
13: $B \leftarrow B \cup$ SLO_AWARE_SCHEDULE(R_{on}, Q_{on}, t, c, m)
14: **if** $m < M_{off}$ **then**
15: PREEMPT_OFFLINE(R_{off}, m, M_{off})
16: $m \leftarrow m - M_{off}$
17: $B \leftarrow B \cup$ SLO_AWARE_SCHEDULE(R_{off}, Q_{off}, t, c, m)
18: Q_{recv}.send(B, t, m)
19: **function** SCHEDULER
20: **Output:** batched requests B
21: scheduled requests B, latency budget t, memory budget $m \leftarrow Q_{recv}$.get(block=True)
22: **if** REQUEST_ARRIVAL **then**
23: UPDATE_BUDGET(B, t, m)
24: **if** $t < 0$ or $m < 0$ **then**
25: PREEMPT_UNTIL_FIT(B, t, m)
26: Q_{send}.send(B)
27: **return** B

A.2 SLO-Aware Prefix Sharing Maximization Algorithm

This section details the SLO-aware prefix sharing maximization algorithm design in HyGen. We construct a prefix tree T_p to capture prefix sharing characteristics among offline requests. During scheduling, offline requests are selected in the DFS order of the prefix tree, and deleted once being scheduled. Additionally, running requests keep their original DFS order in future scheduling process, effectively utilizing prefix sharing.

Algorithm 3 Prefix-sharing-aware offline scheduler

Construct prefix tree T_p
function PREFIX_SHARING_OFFLINE_SCHEDULE
  Input: running requests R, latency budget t, remaining chunk size c, memory budget m
  Output: batched requests B
  $B \leftarrow \{\}$
  for $r \in R$ **do**
    if r.state == DECODE **then**
      $t_{req} \leftarrow$ PREDICTOR.predict(r, DECODE)
      if $t > t_{req}$ **then**
        break
      $t \leftarrow t - t_{req}$
      $B \leftarrow B \cup \{(r, 0, t_{req})\}$
    else if r.state == PREFILL **then**
      $l, t_{req} \leftarrow$ PREDICTOR.get_max_prefill(t, c, m, r)
      if $l > 0$ **then**
        $t \leftarrow t - t_{req}$
        $c \leftarrow c - l$
        $m \leftarrow m -$ GET_NUM_BLOCKS(l)
        $B \leftarrow B \cup \{(r, l, t_{req})\}$
      else
        break
  while T_p is not empty **do**
    $r \leftarrow T_p$.get_next_request()
    $l, t_{req} \leftarrow$ PREDICTOR.get_max_prefill(t, c, m, r)
    if $l > 0$ **then**
      $t \leftarrow t - t_{req}$
      $c \leftarrow c - l$
      $m \leftarrow m -$ GET_NUM_BLOCKS(l)
      $B \leftarrow B \cup \{(r, l, t_{req})\}$
      T_p.remove_request(r)
    else
      break
  return B

A.3 Extended SLO-Aware Prefix Sharing Maximization Algorithm

This section details the extended version of the SLO-aware prefix sharing maximization algorithm, enhanced with fairness-aware scheduling. For offline requests, we construct a prefix tree T_p for prefix sharing, and a self-balancing BST T_f for request freshness. A utility ratio between 0 and 1 is used to balance the chance between these two data structures. During scheduling, the extended prefix sharing maximization algorithm retrieves waiting offline requests from either T_p or T_f, based on the utility ratio. The selected request is then deleted from both data structures to ensure synchronization. This solution balances prefix sharing and fairness without disrupting the DFS order of the prefix tree while avoiding possible starvation.

Algorithm 4 Prefix-sharing-aware offline scheduler

Construct prefix tree T_p and self-balanced BST T_f
function PREFIX_SHARING_OFFLINE_SCHEDULE
  Input: running requests R, latency budget t, remaining chunk size c, memory budget m, utility value u
  Output: batched requests B
  $B \leftarrow \{\}$
  for $r \in R$ **do**
    if r.state == DECODE **then**
      $t_{req} \leftarrow$ PREDICTOR.predict(r, DECODE)
      if $t > t_{req}$ **then**
        break
      $t \leftarrow t - t_{req}$, $B \leftarrow B \cup \{(r, 0, t_{req})\}$
    else if r.state == PREFILL **then**
      $l, t_{req} \leftarrow$ PREDICTOR.get_max_prefill(t, c, m, r)
      if $l > 0$ **then**
        $t \leftarrow t - t_{req}$, $c \leftarrow c - l$, $m \leftarrow m-$ GET_NUM_BLOCKS(l), $B \leftarrow B \cup \{(r, l, t_{req})\}$
      else
        break
  while T_p is not empty **do**
    $rand \leftarrow$ RANDOM_NUMBER(0, 1)
    if $rand < u$ **then**
      $r \leftarrow T_p$.get_next_request()
    else
      $r \leftarrow T_f$.get_next_request()
    $l, t_{req} \leftarrow$ PREDICTOR.get_max_prefill(t, c, m, r)
    if $l > 0$ **then**
      $t \leftarrow t - t_{req}$, $c \leftarrow c - l$, $m \leftarrow m-$ GET_NUM_BLOCKS(l), $B \leftarrow B \cup \{(r, l, t_{req})\}$
      T_p.remove_request(r)
      T_f.remove_request(r)
    else
      break
  return B

A.4 Complexity Analysis of the Two-Phase Scheduling Algorithm

In this section, we present an analysis of the computational complexity of HyGen's two-phase scheduler. The time complexity of its core components is as follows:

- Latency Prediction: $O(1)$ inference using a pre-trained LR model.
- PSM: The initial construction of the prefix tree is $O(NL)$, where N is the number of offline requests and L is the average number of tokens. Each insertion or deletion costs $O(L)$. In implementation, getting the next request in the DFS order only takes $O(1)$, since the DFS order is put in a pre-processed list derived from the prefix tree and can be synced up with the prefix tree asynchronously.
- PSM with Fairness: In the fairness-aware PSM algorithm, a self-balancing BST is used for picking the stalest request. Each lookup, insertion, or deletion takes $O(\log n)$. In implementation, the requests can be kept in an FCFS queue, which syncs up with the BST asynchronously to guarantee correctness, so that each lookup still only takes $O(1)$ time.

Overall, for the two-phase scheduling using existing policies, the time complexity remains $O(n)$ as asynchronous updates can be performed for advanced policies like fairness-aware PSM.

B Further Discussion of the Latency Predictor

Effective co-location in HyGen hinges on a latency predictor that is both highly accurate and computationally lightweight, as it enables real-time scheduling decisions without introducing significant overhead. To this end, we employ a linear regression (LR) model, which provides inference in constant time, $O(1)$, making it ideal for a dynamic serving environment. This section details the model's formulation and its adaptability to the complexities of real-world deployment scenarios.

Model Formulation. The execution time of a serving batch, T_{batch}, is primarily determined by the computational patterns of its prefill and decode stages. We model this relationship as a function of the batch's composition:

$$T_{\text{batch}} = f(S_p, S_d, S_p^2, N_p, N_d) \tag{2}$$

where S_p and S_d represent the total number of tokens in the prefill and decode phases, respectively, and N_p and N_d are the corresponding request counts. The quadratic term S_p^2 is crucial for capturing the non-linear scaling of the self-attention mechanism, which dominates the computational cost of the prefill stage. In contrast, the decode stage, which processes one token per request at a time, exhibits linear scaling with the batch size (N_d). The model's coefficients are pre-trained on data gathered by systematically profiling the target hardware and LLM across a diverse set of batch compositions, a lightweight process that ensures applicability to any deployment environment.

Robustness Through System-Level Design. While a linear model offers unparalleled efficiency, its accuracy in the face of dynamic system conditions (e.g., GPU temperature variations, resource contention) is a critical consideration. HyGen ensures robustness not through a more complex model, but through a synergistic system design. The SLO-aware profiler (Section 4.2) first establishes a macro-level operational latency budget, which implicitly captures the system's current performance characteristics. The LR predictor then operates at a micro-level, making fine-grained decisions on batch composition *within* this pre-calibrated budget.

This two-level approach effectively decouples the real-time scheduling decision from low-level hardware variability. As shown in Figure 16, this design allows the system to maintain robust performance and meet SLOs even in scenarios with predictor error rates exceeding 20%. Furthermore, the feature set is extensible; environmental factors like hardware load can be readily integrated into the model if required for specific use cases.

Adaptability to Modern Model Architectures. The feature set defined in Equation 2 is sufficiently general to generalize across various modern LLM architectures without modification. For instance, for Mixture-of-Experts (MoE) models [48], where a fixed number of experts are activated per token, the resulting computational cost scales linearly with the total number of tokens processed and is effectively modeled by the S_p and S_d features. Similarly, in hybrid architectures that combine

linear-complexity components (e.g., Mamba [19]) with quadratic-complexity attention, our model naturally captures both computational patterns; the linear cost is reflected in the learned coefficient for S_p, while the quadratic cost of the Transformer blocks is captured by the S_p^2 term. This adaptability underscores the predictor's robust design, ensuring its relevance as LLM architectures continue to evolve.

C Taming the Throughput-Latency Tradeoff in LLM Serving with HyGen

Traditional instance scaling solutions address bursty workloads by launching new instances, which can cause tens of seconds to several minutes cold-start delays [15, 64, 66]. To handle these delays, providers often keep standby instances online, leading to wasted resources during off-peak periods.

In contrast, HyGen optimizes resource utilization by running offline workloads on idle resources and reallocating them to online requests in real-time, within a single inference iteration. This eliminates cold-start delays while ensuring high resource utilization.

HyGen complements instance scaling solutions by automating the transition between online and offline workloads, reducing the need for manual intervention. While instance scaling manages large load fluctuations, HyGen ensures efficient resource use during low-traffic periods, optimizing overall system performance in fixed-size clusters.