SkyLB: A Locality-Aware Cross-Region Load Balancer for LLM Inference

Tian Xia[†] Ziming Mao[†] Jamison Kerney[†] Ethan J. Jackson[†‡] Zhifei Li[†§]
Jiarong Xing[†¶] Scott Shenker[†⋄] Ion Stoica[†]
[†]*UC Berkeley* [‡]*Aviatrix* [§]*Renmin University of China* [¶]*Rice University* [⋄]*ICSI*

Abstract

Serving Large Language Models (LLMs) efficiently in multi-region setups remains a challenge. Due to cost and GPU availability concerns, providers typically deploy LLMs in multiple regions using instance with long-term commitments, like reserved instances or on-premise clusters, which are often underutilized due to their region-local traffic handling and diurnal traffic variance. In this paper, we introduce `SkyLB`, a locality-aware multi-region load balancer for LLM inference that aggregates regional diurnal patterns through cross-region traffic handling. By doing so, `SkyLB` enables providers to reserve instances based on expected *global* demand, rather than peak demand in each individual region. Meanwhile, `SkyLB` preserves KV-Cache locality and a balanced load, ensuring cost efficiency without sacrificing performance. `SkyLB` achieves this with a cache-aware cross-region traffic handler and a selective pushing load balancing mechanism based on checking pending requests. Our evaluation on real-world workloads shows that it achieves 1.12-2.06× higher throughput and 1.74-6.30× lower latency compared to existing load balancers, while reducing total serving cost by 25%.

1 Introduction

Large Language Models (LLMs) have seen rapid growth in usage in recent years. With increasingly advanced capabilities, they have been adopted across a wide range of domains, including virtual assistants [8, 27], code generation [16, 23, 48, 52, 55, 56], and information search [15, 45]. These applications now serve billions of users worldwide [46].

Serving LLMs at this scale requires operators to manage GPU scarcity while optimizing cost, latency, and throughput. To meet these demands, LLM providers often deploy infrastructure across multiple geographical regions. The most common practice today is to deploy GPU instances with long-term commitments in each region, like reserved instances [25] or on-premise clusters [11, 49, 50], as shown in Figure 1(a). The long-term commitments reduce GPU costs compared to purchasing on-demand instances with autoscaling (§2.1) and ensure GPU availability at any time, while the region-local deployment improves service quality by placing compute resources closer to end users.

However, this approach introduces significant provisioning and cost management challenges. These reserved instances or on-premise clusters are inflexible—providers cannot increase or decrease capacity within a region as demand shifts. As

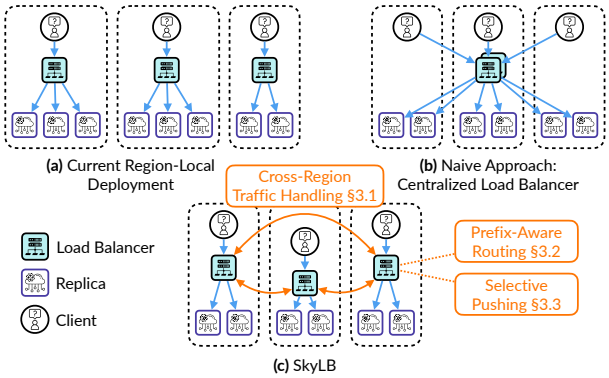


Figure 1. (a) Current region-local deployment is cost inefficient due to provisioning each region for its peak load. (b) A naive centralized load balancer can reduce costs, but it introduces high cross-region latency, performance bottlenecks, and single point of failure. (c) `SkyLB` reduce multi-region serving costs by enabling cross-region traffic handling without sacrificing performance.

a result, providers must allocate enough instances in each region to meet peak demand, which can result in high and often wasted costs. This inefficiency arises from shifting regional traffic patterns that follow distinct daily diurnal cycles: each region experiences peak inference load at certain times of day, with lower demand during off-peak hours (Figure 2). The challenge is further exacerbated in multi-region deployments, where providers must provision for peak load in every region independently. This leads to resource fragmentation and underutilization during off-peak hours.

Ideally, providers would perfectly meet capacity with demand while paying the lower costs of reserved instances or on-premise clusters. However, achieving this ideal is difficult when regional provisioning is viewed in isolation. Providers must either provision for peak demand for all regions or pay for the flexibility of on-demand instances and take the risk of GPU unavailability.

In this paper, we suggest relaxing the regional rigidity of current approaches. Instead of attempting to match regional capacity to regional demand, providers make reservations for peak *global* demand and partition those reservations across the regions closest to their users. Following this approach, when a region is overloaded, it can offload requests to other regions with excess capacity. By reserving for global peak demand and enabling cross-region traffic handling, a system

can improve GPU utilization and reduce overall serving costs. Unfortunately, this cannot be achieved by simply deploying centralized load balancers in a single zone (Figure 1(b), [1, 51]), as this introduces high latency due to cross-region communication and creates both a performance bottleneck and a single point of failure.

It is important to note that cross-region traffic handling, as proposed in this work, has not been a common practice for traditional workloads such as webpage rendering or search engine queries. In these cases, the processing time per request is typically very short [40], often just several tens of milliseconds, and usually smaller than the cross-region network latency that is up to 200 milliseconds [34]. As a result, requests are handled entirely within the region closest to the client. In contrast, LLM requests often require seconds, if not tens of seconds to complete [63]. In this setting, cross-region latency represents only a small fraction of the total processing time. However, responsiveness remains important, and LLM providers still focus on optimizing Time-to-First-Token (TTFT) latency and aim to serve requests locally *when capacity allows*.

To enable cross-region traffic handling without sacrificing performance, we design `SkyLB`, a cross-region load balancer for LLM inference. As shown in Figure 1(c), `SkyLB` deploys at least one load balancer in each region as the first point of contact for requests, ensuring low-latency and avoiding centralized bottlenecks. These regional load balancers collaboratively coordinate traffic across regions to handle load imbalances. Achieving this requires us to take into consideration two key challenges in multi-region load balancing: Key-Value Cache (KV Cache) awareness and LLM inference load unpredictability. We discuss both briefly below before expanding in the rest of the paper.

KV Cache awareness. Modern LLM inference relies heavily on a KV Cache to reuse computation when requests share a common prefix. Previous work has demonstrated that routing such requests to the same GPU is critical for maximizing KV Cache utilization and achieving high throughput [22, 32, 51, 68]. However, these approaches have primarily assumed single load balancer deployments within a single zone. `SkyLB` overcomes the challenge of coordinating prefix-aware routing across multiple regions. It offers two routing algorithms to preserve KV Cache locality in a geo-distributed setting: a simple multi-region extension of consistent hashing based on user ID and session ID, and a more general multi-region prefix trie.

LLM inference load unpredictability. We observe that the processing time and resource usage for LLM inference of a particular request is highly variable and unpredictable [57, 63, 69]. A single request can complete within seconds that requiring very little resources, or takes tens of seconds to complete while requiring several gigabytes of GPU memory. To make matters worse, due to the auto-regressive nature of LLM decoding, it is hard to predict which end of this spectrum a particular request will fall on [9]. As we show in section §2.3, this property of LLMs makes traditional load balancing policies such as least load first or round robin highly ineffective. To address this, `SkyLB` implements selective pushing based on pending requests, an algorithm we describe in section §3.3, which improves performance by balancing load based on each replica's availability of admitting more requests in its continuous batch [62].

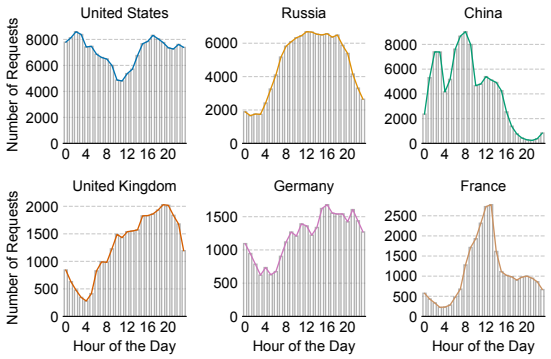


Figure 2. Regional traffic demands shift over time in multi-region LLM serving. Data is from the WildChat [65] trace.

This paper proceeds as follows. In section §2, we describe the background of LLM serving, motivate the need for cross-region routing to lower cost, and identify new challenges in effectively doing so. In §3, we introduce the key design choices of `SkyLB`, including cross-region traffic handling, multi-region prefix-aware routing, and selective pushing to mitigate load imbalance. We evaluate `SkyLB` on three real-world workloads, and we observe that `SkyLB` achieves 1.12-2.06× higher throughput and 1.74-6.30× lower latency compared to existing load balancers. By using cross-region routing, we show that `SkyLB` is able to reduce 25% of total cost compared to region-local deployment. In summary, this paper makes four contributions:

- Identifying the need for cross-region traffic handling to aggregate diurnal patterns across multiple geographical regions and reduce global serving cost.
- Proposing two mechanisms to provide effective cross-region routing: prefix-aware routing to improve cache locality and performance, and selective pushing based on pending requests at each replica to reduce load imbalance.
- A comprehensive evaluation for `SkyLB`, compared to existing production and research systems across a variety of workloads.
- An open-source system `SkyLB` to stimulate further research on cross-region load balancing for LLMs.

2 Background and Motivation

In this section, we first provide background on multi-region LLM serving. We then introduce the global cost reduction

problem, which motivates cross-region load balancing. Finally, we discuss the key challenges in enabling cross-region load balancing without sacrificing performance.

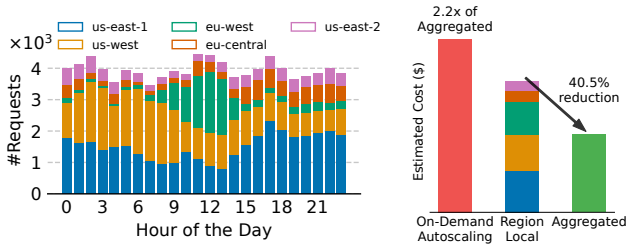
2.1 Background: Multi-Region LLM Serving

LLM inference. LLM inference typically consists of two stages: prefill and decode. During the prefill stage, the model processes the initial input prompt and generates an internal KV Cache, which stores intermediate states necessary for subsequent token generation. Following prefill, the decode stage generates tokens auto-regressively, one token at a time, utilizing the KV Cache to accelerate inference. To improve GPU utilization and throughput, continuous batching [62] is commonly employed, which dynamically groups incoming requests to reduce idle time.

Scaling LLM serving to multiple regions. To scale effectively, major providers [38] leverage GPU resources across multiple geographical regions to serve their users, a practice we refer to as *multi-region serving* in this paper. Multi-region serving offers two key benefits: (1) improved latency by deploying LLM replicas closer to users; (2) reduces the risk of GPU shortages in any single region by diversifying GPU usage across multiple regions [34, 60].

Provisioning GPUs via long-term commitments. The substantial GPU demands of LLM serving, combined with ongoing GPU scarcity, have led providers to primarily deploy reserved or on-premise GPU instances. Such long-term GPU commitments not only ensure GPU availability at all times but also offer lower prices over extended periods compared to on-demand instances. For example, a three-year reserved instance for 8 `H100` GPUs (`p5.48xlarge`) on AWS costs $37.56/hour, while the equivalent on-demand instance costs $98.32/hour. On-premise deployments can offer even greater cost savings: as shown in [3], they can reduce costs by up to 46.3% over time compared to reserved cloud instances when accounting for lifetime return on investment. As a result, this strategy has become the norm among today's LLM service providers. For example, OpenAI has deployed tens of thousands of GPUs in its data centers [38, 49]; similar deployments are also reported by ByteDance [50] and xAI [11].

Latency metric: Time-to-First-Token (TTFT). A critical performance metric for online serving is TTFT, defined as the latency from when a request initiates until the first output token is produced. When the client and model replica are in the same region, TTFT primarily consists of prefill latency and queuing delay. For example, when deploying on a L4 GPU, processing a 512-token prompt with `meta-llama/Llama-3.1-8B-Instruct` might incur around 300 ms of prefill latency before generating the first token. In a multi-region deployment, TTFT additionally includes cross-region network latency, which typically adds up to 200 ms.



(a) Regional load. **(b)** Cost comparison.

Figure 3. (a) Aggregated load across five regions using a subset of the WildChat trace [65]. Before aggregation, per-region load variance ranges from 2.88× to 32.64×; after aggregation, the variance is reduced to 1.29×. (b) Cost reduction achieved by provisioning based on the aggregated *global* peak load, using reserved instances. We also present the cost for perfect on-demand autoscaling, assuming precise traffic prediction and no provisioning delay. In practical scenarios, the actual cost of on-demand autoscaling would be even higher.

2.2 New Problem: Global Cost Reduction

Multi-region LLM serving introduces new challenges for reducing serving costs. Our trace analysis reveals that multi-region LLM serving exhibits regional demand shifts over time, often following a diurnal pattern. Specifically, Figure 2 illustrates the load patterns of six countries from the WildChat trace [65]. It shows clear diurnal trends—each region experiences peak inference traffic during specific hours, with lower load during the rest. These peak hours vary across regions due to time zone differences.

The fluctuating load leads to time-varying GPU demand in each region, posing significant challenges for resource provisioning and cost reduction. Ideally, providers would provision just enough GPU resources in each region to meet the required GPU demand and adjust allocations dynamically as load varies. Recent studies attempt to approach this ideal by auto-scaling resources using on-demand or spot instances [34, 36, 44, 64]. However, as discussed in Section §2.1, to avoid GPU shortages during demand spikes and to secure lower instance pricing, current multi-region deployments typically rely on static provisioning via long-term commitments. This makes them unable to scale flexibly with load fluctuations. As a result, to maintain service quality, providers must provision each region for its peak load. While this approach ensures low latency and high throughput, it inevitably leads to low overall GPU utilization and higher operational costs.

To illustrate this, Figure 3a shows the aggregated number of requests received across five regions at different hours of the day, using a subset of the WildChat trace [65]. It highlights that while regional loads fluctuate significantly, the aggregated global load remains relatively stable. As shown in Figure 3b, provisioning based on the peak aggregated global load can reduce costs by 40.5% compared to provisioning

each region for its own peak independently. With on-premise instances, the cost can be reduced even further compared to reserved instances [3]. The figure also includes the cost of using on-demand instances. Due to their high pricing, even with perfect auto-scaling, where we assuming no provisioning delay and instances are always available, this approach incurs 2.2× the cost compared to provisioning the aggregated global load using reserved instances.

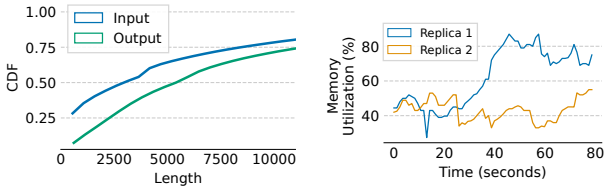
Insight. The cost reduction shown in Figure 3b suggests that we should provision instances based on *global* demand and share these resources across regions. Users can be routed to access their geographically closest region, but when it becomes overloaded, traffic can be rerouted to remote regions to utilize otherwise unused capacity.

2.3 Cross-Region Load Balancing

Achieving global cost reduction requires cross-region load balancing, a direction that remains largely unexplored in the context of LLM serving. Today's LLM load balancers are designed for single-region deployments and cannot be directly extended to multi-region settings. As shown in Figure 1(b), a straightforward approach is to deploy existing load balancers in a specific zone as global coordinators. However, this creates centralized bottlenecks and a single point of failure, while also incurring non-trivial latency overhead due to cross-region communication for most requests. In typical local deployments, the TTFT latency for LLMs is mainly bottlenecked by prefill time and is usually several hundred milliseconds [54], while cross-region latency can reach up to 200 milliseconds [34]. This means that the straightforward approach may introduce additional latency nearly equivalent to the prefill time, significantly impacting responsiveness. This motivates us to design a geo-distributed load balancer for LLM serving that can effectively reduce global serving costs without compromising TTFT. This involves addressing two key challenges:

KV Cache awareness. LLM serving systems reuse KV Cache between requests that share the same prefix to reduce prefill time and saving compute resources. To maximize the KV Cache hit rate, prior work has explored making load balancers prefix aware [1, 12, 51]. These approaches typically maintain a prefix tree for each replica, either precise, as in Preble [51], or approximate, as in SGLang Router [1] and DLPM [12]. This line of work has shown strong potential, and we observed up to a 35% improvement in KV Cache hit rate in the best case.

However, extending them to multi-region deployments presents several challenges. Achieving KV Cache awareness requires maintaining a shared prefix tree, which is difficult to coordinate across regions. As previously discussed, simply maintaining a centralized global prefix tree will incur significant latency penalties due to frequent remote accesses. Alternatively, maintaining fully distributed prefix trees in each region requires cross-region coordination on every request, which would become prohibitively expensive due to the high coordination overhead.



(a) CDF for request length. **(b)** Load imbalance.

Figure 4. (a) The CDF of input length and output length in WildChat dataset. (b) We observe load imbalance across replicas when routing requests using the Round Robin algorithm. The y-axis shows the percentage of KV Cache memory utilization on each replica. The peak memory usage difference between replicas reaches 2.64×.

LLM inference load unpredictability. The length of LLM outputs can vary widely, and in some cases be very long, as shown in real workloads (Figure 4a) and prior studies [57]. This variability, combined with the auto-regressive nature of LLMs, makes it difficult for load balancers to predict output lengths in advance and accurately estimate the GPU resources required for each request. Moreover, each LLM request can demand substantial resources. It is not uncommon for a single request to consume several gigabytes of GPU memory, limiting each model replica to handling only tens of concurrent requests. Traditional load balancing polices, such as least-load-first or round-robin, blindly push requests to replicas without accounting for resource consumption, often resulting in long-running requests blocking all subsequent ones in the queue and causing severe load imbalance. These unique characteristics of the LLM workload make the *penalty of misrouting* much more severe than in traditional CPU-based workloads. This motivates the need for a new system design that is robust to the dynamic and resource-intensive nature of LLM workloads.

3 `SkyLB` Design

Overview. In this work, we introduce `SkyLB`, a new design that enables efficient cross-region load balancing for LLM inference. With `SkyLB`, LLM service providers can reduce global costs by sharing a smaller pool of reserved or on-premise instances across regions without sacrificing performance. `SkyLB` deploys load balancers in multiple regions as the first point of contact for local requests, and introduces a cross-region traffic handler that coordinates traffic between regional load balancers to mitigate cross-region load imbalance (§3.1). It preserves the benefits of prefix sharing by supporting prefix-aware routing in two ways (§3.2): (1) a simple yet effective policy based on consistent hashing that requires

minimal changes to existing load balancers; and (2) prefix-aware routing using partial prefix tree snapshots maintained at each load balancer. In addition, to address the challenge of LLM inference load unpredictability, `SkyLB` introduces a novel selective pushing mechanism that balances load based on pending requests at each replica (§3.3). We detail each of these design choices in the following.

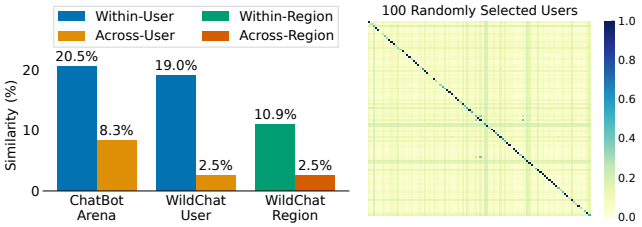
3.1 Cross-Region Traffic Handling

Since geographical regions could experience peak load at different times, we can exploit the traffic pattern by offloading workloads from high-demand regions to low-demand ones. This mitigates the load imbalance caused by regional demand shifts over time and reduces the total number of required GPU instances compared to region-local deployment.

There are multiple possible approaches to handling cross-region traffic. One approach is to deploy load balancers in a single zone that manage replicas across all regions. In this setup, requests from all regions are first sent to the central load balancer, which then routes them to replicas possibly in different regions. However, as discussed in §2.3, this results in long round-trip times for users whose requests traverse multiple regions: the request incurs two cross-region RTTs, i.e., one to reach the load balancer and another to reach the assigned replica, leading to high cross-region latency.

Another approach is to replicate the load balancer across multiple regions, allowing each to route traffic to all available replicas. In this deployment, clients send their requests to the nearest load balancer, which then decides which replica should handle the request, potentially in any region. However, this approach requires non-trivial synchronization *between load balancers* to make coordinated routing decisions. Without such coordination, multiple load balancers may independently select the same replica as hot spot, leading to degraded performance, particularly for affinity-aware load balancing strategies. Such coordination can incur non-trivial overhead for each load balancer, requiring $O(N_{\text{LB}} \times N_{\text{replica}})$ connections or probes. Whenever a new replica is launched, it must ensure that all load balancers are aware of its creation and updated status, adding complexity and latency to the system.

Our approach: two-layer cross-region routing. Instead, we adopt a two-layer approach. The key idea is to coordinate cross-region traffic between load balancers, rather than directly between replicas. Specifically, each load balancer either routes requests to local replicas or forwards them to other load balancers in remote regions, which then make the final placement decisions within their region. This design combines the strengths of the two approaches discussed earlier. It avoids introducing significant routing latency by allowing clients to connect to their nearest load balancer, while still enabling cross-region load balancing through coordination among load balancers. Comparing load balancers routing to all replicas distributed across regions, this approach significantly reduces the probing overhead required to assess replica load. In addition, managing connections between a small number of load balancers scales much better than maintaining connections to every replica, as the number of replicas typically far exceeds the number of load balancers.



(a) Prefix similarity. **(b)** Similarity heatmap.

Figure 5. (a) Average prefix similarity within and across users and regions; (b) Heatmap of pairwise prefix similarity among 100 randomly sampled users, in WildChat [65] and ChatBot Arena [67] datasets. User information is retrieved directly from the metadata provided in each dataset (hashed IP in WildChat and judge in ChatBot Arena).

3.2 Multi-Region Prefix-Aware Routing

The need for cross-region traffic routing presents new challenges for prefix-aware routing (§2.3), especially with long cross-region latency. Achieving optimal prefix-aware routing requires a *global* view of prefix states across all replicas. However, each load balancer only observes a *subset* of requests, making it difficult to maintain a consistent global view without incurring significant coordination overhead on every request. Thus, we ask the question: How can we effectively load balance across *multiple regions* with prefix awareness? We answer this question by first understanding prefix sharing patterns in real workloads.

Prefix similarity analysis. We analyze prefix similarity[1] using a subset of the WildChat [65] and ChatBot Arena [67] datasets. The goal is to quantify how much prefix reuse occurs in real-world workloads, which directly impacts the effectiveness of KV Cache reuse in LLM serving systems. This metric measures the normalized length of the common prefix shared between two requests. We compute prefix similarity across all pairs of requests within the same user, and across different users. The results are shown in Figure 5a. We observe that the average prefix similarity within the same user is significantly higher than that across different users (2.47-7.60× more). This pattern is also evident in the heatmap (Figure 5b), which shows the similarity among 100 randomly selected users, further confirming that within-user requests are more likely to

[1]We define the prefix similarity between two requests a and b as $\text{len}(\text{common\_prefix}(a,b))/\min(\text{len}(a),\text{len}(b))$. We use the minimum length in the denominator so that, for example, if a is a prefix of b, the prefix similarity of a and b should be 1.

share context and thus benefit from prefix caching. However, there is still some degree of cross-user prefix similarity, and the relative ratio between within-user and cross-user prefix similarity is workload-dependent (2.47× for ChatBot Arena and 7.60× for WildChat). This observation motivates us to present the following two solutions: SkyLB-CH and SkyLB. SkyLB-CH is simple and captures user-level prefix similarity. SkyLB captures both within-user and cross-user prefix similarity. We detailed the algorithm in Listing 1.

SkyLB-CH. SkyLB-CH uses consistent hashing [29] on user-provided keys (e.g., user ID, session ID) and routes a user request to a corresponding replica (Listing 1, lines 23-26). SkyLB-CH is *implicitly* prefix-aware: requests from the same user tend to share similar prefixes (e.g., context, chat history) and consistent hashing will map them to the same replica. SkyLB-CH adopts a ring hash [53] scheme, where each virtual node on the hash ring is assigned to a replica and each replica can have multiple virtual nodes, allowing balanced key distribution across replicas. We make two extensions to the traditional consistent hashing. First, due to SkyLB's two-layer load balancing design, SkyLB-CH performs consistent hashing at both layers: the load balancer routes requests to other balancers based on consistent hashing, and each balancer applies consistent hashing to assign the request to one of its managed replicas as well. Second, virtual nodes are skipped based on the availability of its associated replica (detailed in §3.3, and Listing 1, line 26). When that happens, the algorithm continues iterating over successive virtual nodes on the ring. SkyLB-CH requires minimal state maintained at load balancers, and can be easily incorporated into the existing software stack.

Since SkyLB-CH focuses only on within-user prefix similarity, there are cases where SkyLB-CH falls short of being optimal. We discuss them in the following:

- **Cross-User Prefix Sharing**: Requests from different users can share common prefixes (Figure 5), but SkyLB-CH can route these requests to two different replicas, missing the benefits of routing these requests to the same replica to maximize KV Cache hit rate.
- **Bursty Request**: Consistent hashing either hashes a given key to a single replica, or to a replica set. In the former, a request burst can overload the single replica. In the latter, consistent hashing misses the opportunities to leverage prefix-sharing for requests sent by the same user to different replicas in the replica set.
- **Heterogeneous User Program**: If requests from a single user's program contain multiple patterns and lack consistent prefix structures, using the user ID (or session/program ID) as the hash key fails to exploit prefix reuse. Worse, it may route dissimilar requests to the same replica, increasing the risk of overloading that replica.

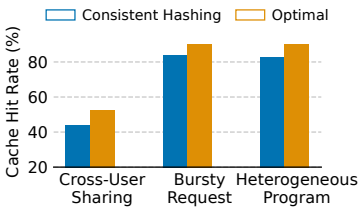


Figure 6. KV Cache hit rate, comparing consistent hashing with optimal solution with a global view.

We illustrate these scenarios in Figure 6, which shows the KV Cache hit rate under each setting, compared to an optimal solution with a global view. The lack of cross-user prefix sharing results in a 16.49% prefix hit rate drop. We also study behaviors under bursty request patterns, where the variance in a single user's concurrent requests can reach 4×. In this case, CH leads to 7.07% lower hit rate. A similar trend is observed when users submit heterogeneous programs, resulting in a 8.78% gap. These limitations motivate us to develop SkyLB that is more general than SkyLB-CH by maintaining more states at the load balancer.

SkyLB with regional snapshot. SkyLB is *explicitly* prefix-aware: in this design, each load balancer maintains prefix trees to keep an approximate view of prefix information on the load balancing targets. Between load balancers, the targets are remote load balancers, and between the load balancer and the replica, the targets are local replicas managed by that load balancer.

The prefix tree is a logical trie augmented with metadata to track active load balancing targets at each node. Each node stores a set of active targets associated with the prefix formed by the path from the root to that node. The tree is built incrementally from the requests the load balancer has served: when a new request is forwarded, a corresponding path is added to the trie, and the selected target is recorded at every node along that path. To bound memory usage, SkyLB enforces a configurable maximum tree size and evicts entries when the tree exceeds this limit, starting with the earliest inserted records. SkyLB filter targets based on whether it is available to serve requests and pick the available target with the longest matching prefix (detailed in §3.3, and Listing 1, line 21). Specifically, for each trie traversal step, if there is no *available* matching load-balancing target in the current node, the traversal terminates early. This is because the set of targets stored in any child node is always a *subset* of its parent's, implying that no available replicas can be found further down the path.

Each load balancer maintains two prefix trees, one for local replicas it manages and one for a partial view (snapshot) of other load balancers in other regions. The latter keeps track of all historical requests that the local region has sent to remote regions. Regional snapshots do not strictly record all prefixes reside in replicas of remote regions, which depends

Algorithm 1 `SkyLB` load balancing logic

```
1:  procedure MonitorAvailability
2:      while true do
3:          for all r ∈ LocalReplicas do
4:              n_pending ← ProbeReplica(r)
5:              if n_pending > 0 then
6:                  Remove(LocalAvail, r)
7:              else
8:                  Add(LocalAvail, r)
9:          for all lb ∈ RemoteLBs do
10:             (n_avail_replica, size_q) ← ProbeLB(lb)
11:                 ▷ τ: small buffer for newly arriving requests
12:             if n_avail_replica = 0 ∨ size_q > τ then
13:                 Remove(RemoteAvail, lb)
14:             else
15:                 Add(RemoteAvail, lb)
16:         Sleep(ProbeInterval)
17: procedure SelectCandidate(Request, C)
18:     if UsePrefixTree then
19:         Text ← GetText(Request)
20:         Trie ← { ReplicaTrie    if C is replicas
                   { LBSnapShotTrie otherwise
21:         return MaxPrefixMatch(Trie, Text, C)
22:     else
23:         HashRing ← { ReplicaRing if C is replicas
                       { LBRing     otherwise
24:         Key ← SessionID(Request)
25:         HashValue ← Hash(Key)
26:         return Next(HashRing, HashValue, C)
27: procedure HandleRequest(Request)
28:     C ← { LocalAvail  if LocalAvail ≠ ∅
            { RemoteAvail otherwise
29:     t ← SelectCandidate(Request, C)
30:     Route(Request, t)
```

on requests that are sent to the load balancer of that region, either directly or from other load balancers. Instead, it is an approximation of prefixes *that are possible to be utilized by local region forwarding to that remote region*, as we observe empirically that the local region is unlikely to share prefixes with requests that came from other regions: in Figure 5a, requests across regions only have 2.5% prefix affinity. With that, we observe `SkyLB` more closely approaches the performance of an optimal solution.

3.3 Selective Pushing to Mitigate Load Imbalance

While leveraging prefix-affinity improves performance, it also leads to load imbalance as requests are preferentially routed to specific replicas. To address this, prefix-aware routing must be combined with effective load balancing strategies—for example, switching to a load-balancing policy when the prefix sharing ratio falls below a certain threshold. However, traditional load balancing strategies, such as blind pushing and selective pushing based on the maximum number of outstanding requests, do not work effectively for LLM workloads due to their load unpredictability (§2.3). We begin by analyzing these two strategies and show how they lead to load imbalance when applied to LLM inference workloads. We then present our approach, selective pushing based on pending requests, as a solution to this problem.

Blind pushing. One traditional load balancing strategy is to route each request to a replica *immediately* upon arrival [1, 17, 51], which we refer to as *blind pushing*. Blind pushing performs well in CPU-based workloads or scenarios with uniform request processing times, where simple strategies like round-robin or least-load-first naturally result in a balanced load. However, the processing time of LLM varies from request to request, as it depends on the output length, which is difficult to predict due to the auto-regressive nature of decoding (§2.3). Naively assuming each request is homogeneous can lead to a significant load differences across replicas. For example, we find two replicas under round robin can have memory usage difference up to 2.64×, as shown in Figure 4b. This issue is especially problematic when routing requests among multiple *overloaded* replicas. A replica with a seemingly short queue may still incur long processing times if the requests in the queue takes long time to process. Blindly pushing requests to such overloaded replicas can lead to cases where requests waiting in one replica's queue while other replicas have idle compute capacity, wasting compute resources.

Selective pushing. To address the unpredictability of LLM inference, we suggest selective pushing, a strategy where requests are temporarily queued at the load balancer and sent only to replicas that meet certain conditions. Specifically, the load balancer will only push requests to the replica that has capacity (decided by a threshold), and in the event that all replicas are full, queuing requests at the load balancer. This approach prevents overloading any single replica while maximizing overall utilization across all available replicas, ensuring that no request waits in one replica's queue while others have idle compute capacity. We explain two thresholds, outstanding requests and pending requests for selective pushing, and show that the latter is preferred in LLM serving.

Selective pushing by limiting outstanding requests. In this method, the load balancer selectively pushes to a replica only when the number of of outstanding requests for that replica is less than a *fixed* threshold [20, 31, 47]. Each replica will not exceed its desired level of load and the rest will be queued in the load balancer. That way, when the request finishes on a replica and releases free capacity, it will inform the load balancer so that new requests are permitted to be served at this replica. However, selective pushing based on a fixed number of outstanding requests is ineffective for LLM workloads, since the number of requests a replica can serve depends on the total memory footprint of all outstanding

requests, which is proportional to the total number of input and output tokens. As the number of output tokens cannot be predicted in advance, the same inference engine can host a small number of large requests or a large number of small requests. We observed that for Llama 3.1 8B on a L4 GPU, the max number of outstanding requests can range from 20 to 50 for the same dataset. Therefore, statically setting the maximum threshold of outstanding requests delivers poor performance for LLM service (§5.2).

Selective pushing by checking pending requests. We propose selective pushing by checking *pending requests*. A *pending request* is a request that has not been scheduled to the continuous batch yet, which indicates that the current batch is full and cannot admit more requests, as constrained by GPU memory. We use *the existence of* pending requests in the replica to decide whether a replica is full or not. A background heartbeat probe is periodically sent to replicas to obtain their pending queue size (Listing 1, line 3-8). If a replica has no pending request, it is ready to serve more requests.

Selective pushing and cross-region routing. Each load balancer tracks the number of replicas it manages with full continuous batches and periodically synchronizes this state with peer load balancers through heartbeat messages (Listing 1, line 9-15). If a load balancer has at least one non-full replica and its request queue size does not exceed a small buffer (line 12), it is considered available to accept additional requests. Whether a peer load balanacer is available to serve requests is used to guide cross-region routing. When at least one local replica is not full, requests are always routed locally to maximize responsiveness. If all local replicas are full, the system considers remote regions and forwards requests only to regions with available replicas and short load balancer queue (Listing 1, line 28). When multiple candidates are available, either among local replicas or remote load balancers, the system breaks ties using the consistent hashing key (for `SkyLB`-CH) or the prefix hit rate (for `SkyLB`) to select a candidate with more prefix sharing, as detailed in Listing 1, line 17-26.

4 `SkyLB` Implementation

We implemented `SkyLB` (Figure 7), a prototype system that leverages geo-distributed load balancers to achieve both high throughput and low latency for online LLM serving across multiple geographical regions. `SkyLB` is built on top of SkyServe [34], an open-source multi-region serving framework for AI models, which supports both on-premise and cloud-based replicas. `SkyLB` extends SkyServe by adding geo-distributed load balancer with ≈ 3000 lines of Python code, and is compatible with any inference engine with OpenAI API, such as vLLM [32] and SGLang [68].

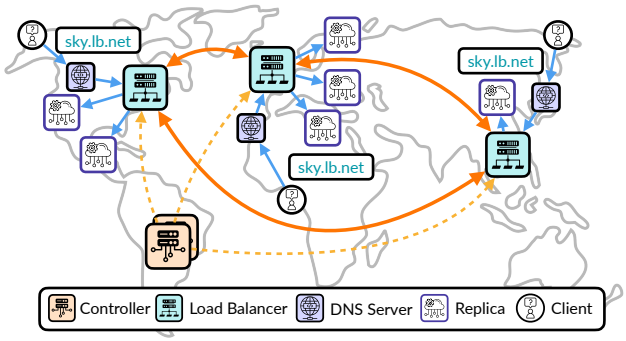


Figure 7. System architecture of `SkyLB`. Clients across different geographical regions issue requests to a unified domain name (`sky.lb.net` in this example). The domain will be resolved to the nearest available load balancer based on the client's IP. Each load balancer maintains connections to local replicas and other remote load balancers and performs request routing based on its load and shared prefix (§3.2). Load balancers coordinate with each other to handle load redistribution. A centralized controller manages system updates, monitors health via periodic probes, and orchestrates failure recovery across all load balancers and replicas. Note that the placement of load balancers and replicas in the figure is illustrative; their geographic positions are abstracted for clarity and do not reflect exact deployment coordinates.

4.1 Load Balancer

Load balancer distribution. `SkyLB` launches load balancers in user-specified regions, with the number of instances configurable by the user. In cloud deployments, the regions are automatically inferred from the cloud region. `SkyLB` creates an AWS Route53 [6] DNS record for each load balancer, all associated with the same domain name for a unified endpoint. Under the hood, DNS resolution maps the domain name to the nearest load balancer based on the client's source IP, thereby routing client requests to the nearest one.

Request life cycle. A request first contacts the DNS server to resolve the IP address of the nearest load balancer. It is then sent to the load balancer and placed in a first-come, first-served (FCFS) request queue. When the request reaches the head of the queue, *available* replicas in the local region are prioritized. If no local replica is available, the request is forwarded to an *available* remote region with the highest prefix hit ratio. If the request is routed to a remote region, its snapshot is updated using the input prompt of this request.

4.2 Serving System

Service controller. `SkyLB` employs a centralized controller to manage any updates to the system, such as adding or removing replicas and reconfiguring load balancers. It also performs periodic probes to monitor the status and health of all replicas

and load balancers. The controller is fault-tolerant and can recover its state automatically in the event of a failure.

Failure recovery of load balancers. SkyLB supports automatic recovery for load balancers. Upon detecting an unexpected failure via periodic health probes, the controller reassigns the replicas in the affected region to a geographically closest load balancer. That load balancer will then temporarily treat those replicas as local replicas. In parallel, a recovery process is initiated in the background. Once the failed load balancer is recovered, the associated replicas are transferred back to it. SkyLB can tolerate multiple concurrent load balancer failures. For higher fault tolerance, users can deploy additional load balancers in any region.

5 Evaluation

We evaluate SkyLB comprehensively across a variety of workloads and configurations to answer three questions:

- Can SkyLB maintain high throughput while preserving low latency in a geo-distributed setup? (§5.1)
- What performance gains does selective pushing with pending requests (§3.3) provide? (§5.2)
- What performance and cost benefits does SkyLB provides for regionally imbalanced workloads compared to standard region-local deployments? (§5.2)

5.1 Macrobenchmarks

We conducted end-to-end experiments using up to 12 replicas in a multi-region setup, where both replicas and clients are distributed across three geographical regions. We compare SkyLB with several production and research systems:

- **GKE Gateway [24]**: GKE Gateway is a network gateway service that connects multiple GKE [2] clusters to provide a unified endpoint. Under the hood, each request is routed to and handled by one of the clusters.
- **Round Robin (RR)**: A stateless load balancer that distributes incoming requests in a round-robin fashion.
- **Least Load (LL)**: A load balancer that tracks the number of outstanding requests per replica and routes each new request to the replica with the least load.
- **Consistent Hashing (CH)**: A ring hash [29, 53] based consistent hashing algorithm, using the user's IP address and session ID as hash key.
- **SGLang Router [1] (SGL)**: A prefix-aware load balancer that routes requests based on a cache-aware routing algorithm tailored to LLM workloads.
- **SkyLB-CH**: SkyLB using a ring-hash based consistent hashing policy.
- **SkyLB**: SkyLB using the prefix tree policy.

For the baselines RR, LL, CH, and SGL, we deploy a single load balancer in the US. For both variants of SkyLB and GKE Gateway, a load balancer is deployed in each region.

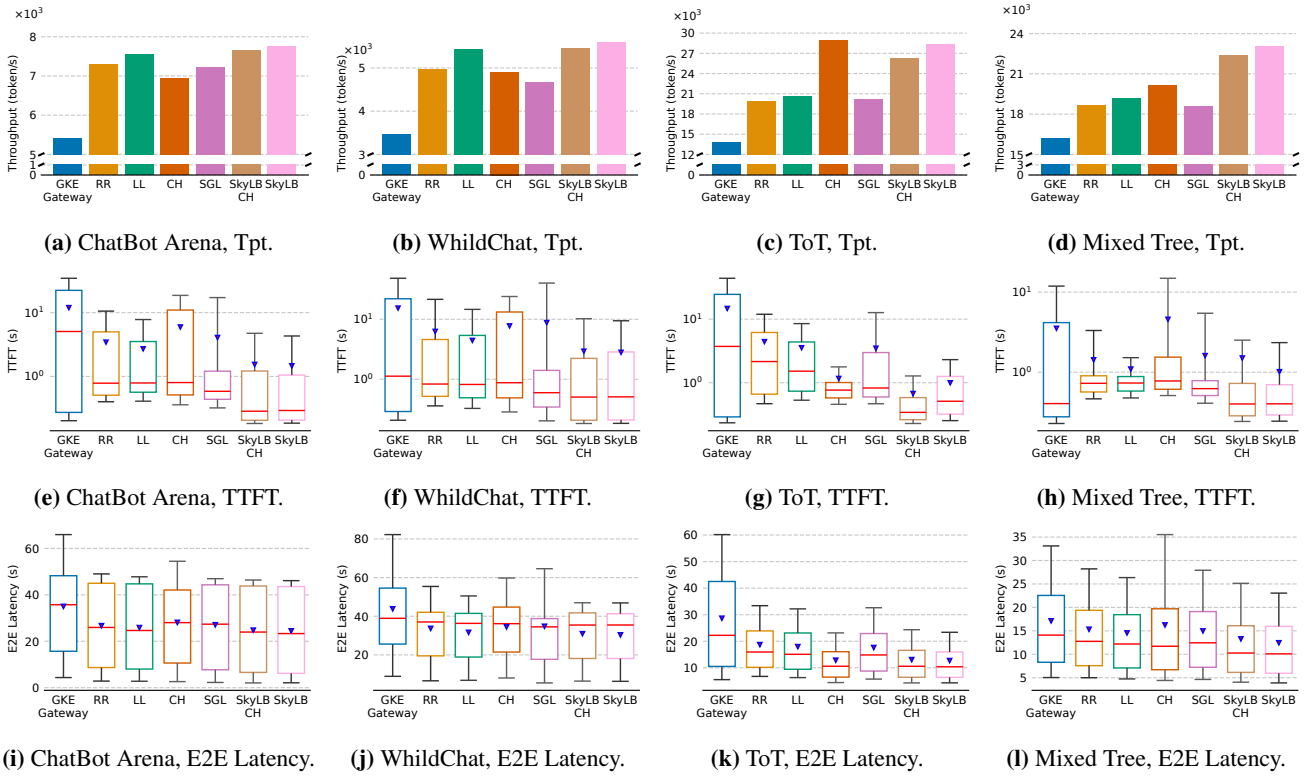
Experiment setup. We conduct our evaluation primarily on AWS [7], except for the GKE Gateway experiments which are performed on GCP [26]. To ensure a fair comparison, each system uses the same replica configuration. All replicas use one L4 GPU, hosting the `meta-llama/Llama-3.1-8B-Instruct` model via SGLang [68]. Replicas are distributed across three regions: the United States, Europe, and Asia. We vary replica's geographical allocation and client workload pattern to test a range of scenarios. For all experiments, we deploy clients in the US, Asia, and Europe to generate traffic, representing all end users in its respective region. Each client issues one program at a time. We use the following workloads:

Multi-turn conversation. We evaluate all systems on several multi-turn conversation datasets and vary the client configuration to reflect different deployment scenarios:

- **ChatBot Arena [34]**: A real-world multi-turn LLM conversation dataset collected using anonymized user IDs. For each region, we maintain the same number of clients, with 80 ongoing conversations per region. The real user ID in the dataset is used as its consistent hashing key.
- **WildChat [65]**: A large dataset of one million multi-turn conversations with demographic metadata such as state, country, and hashed IP address. We evaluate a configuration with different numbers of clients across regions: 40 in the US and 30 in both Europe and Asia. Each region issues requests only for conversations from its own geographical area, defined by the dataset's metadata. The hashed IP in the dataset is used as its consistent hashing key.

Tree of Thoughts. We also evaluate on the Tree of Thoughts [61] benchmark using the Grade School Math dataset [14] from OpenAI. In this setting, the replica configuration is balanced, with 12 replicas evenly distributed across all regions (four per region). Tree of Thoughts exhibits high prefix reuse, as each question is solved via a tree structure where multiple nodes share prefixes from root to their least common ancestors. Nodes in the same tree can be executed concurrently. The tree has a depth of four, corresponding to a multi-step math reasoning task. The question ID in the dataset is used as the consistent hashing key. We evaluate two workload types:

- **Tree of Thoughts (ToT)**: Each tree uses a 2-branch structure (15 requests per tree). The US region runs 40 clients in parallel, while Europe and Asia run 20 clients concurrently.
- **Mixed Tree**: A more complex scenario where US runs 4-branch trees (85 requests per tree), with two clients sending such tree concurrently. The remaining regions continue to issue 2-branch trees, each with 20 clients in parallel. This setup reflects a mixed workload scenario where users generate heterogeneous traffic (e.g. setting different branch sizes for different accuracy requirements), more accurately representing real-world usage patterns.



(a) ChatBot Arena, Tpt. **(b)** WhildChat, Tpt. **(c)** ToT, Tpt. **(d)** Mixed Tree, Tpt.

(e) ChatBot Arena, TTFT. **(f)** WhildChat, TTFT. **(g)** ToT, TTFT. **(h)** Mixed Tree, TTFT.

(i) ChatBot Arena, E2E Latency. **(j)** WhildChat, E2E Latency. **(k)** ToT, E2E Latency. **(l)** Mixed Tree, E2E Latency.

Figure 8. Service Throughput, TTFT Latency, and End-to-End Latency. We run `meta-llama/Llama-3.1-8B-Instruct` on one L4 GPU with up to 12 replicas and report service throughput along with the distributions of TTFT and end-to-end latency. The TTFT latency plot is log-scaled. For the box plot, the red line marks the median, the box marks 25^{th} and 75^{th} percentiles, the whiskers show 10^{th} and 90^{th} percentiles, and the inverted triangle marks the mean.

We report end-to-end service throughput, TTFT latency and end-to-end latency to evaluate system performance and responsiveness (Figure 8).

Service throughput. We show the service throughput of multi-turn conversation datasets (ChatBot Arena and WildChat) in Figrue 8a, 8b. Both variants of `SkyLB` improve service throughput by 1.12-1.2× compared to single load balancer solutions. Prefix-aware baselines such as SGL and CH rely on blind pushing, which leads to overloading some replicas while leaving others underutilized. In these baselines, replicas experience high variance in outstanding request counts, ranging from 2.33-5.08× for SGL and 2.54-4.92× for CH. Non-prefix-aware baselines perform worse in terms of prefix hit rate. RR achieves only 10.78-16.57%, while LL performs better with 28.29-31.13%, though still below `SkyLB` 's higher hit rate of 36.96-46.55%. Among the single load balancer baselines, LL achieves the best throughput, as load balancing plays a more dominant role when prefix similarity is relatively low. Nevertheless, it still falls short of `SkyLB`, reaching 97.38% of `SkyLB` 's throughput. Compared to GKE Gateway, `SkyLB` achieves a throughput improvement of 1.43-1.62×. This gain is primarily due to `SkyLB` 's LLM-specific design. While GKE Gateway offers robust, general-purpose multi-cluster load balancing, it lacks prefix-aware routing for KV Cache optimization and the selective pushing mechanism that adapts to the dynamic nature of LLM workloads. The absence of these capabilities in a standard gateway solution results in lower cache hit rates (18.08-24.30%) and less efficient GPU utilization, thereby constraining overall service throughput.

In the Tree of Thoughts workload (Figrue 8c), when all trees are of uniform size, the CH baseline slightly outperforms `SkyLB` with a marginal throughput gain of 2%. CH also outperforms LL by 1.4× in ToT, due to substantial prefix sharing. CH hashes requests from the same tree (i.e., the same question) to the same replica, enabling effective reuse of cached prefixes. However, this advantage disappears under heterogeneous workloads (e.g., 2-branch vs. 4-branch trees, Figrue 8d), and user-generated request bursts can saturate individual replicas. In such cases, the CH policy continues routing requests from the same user to the same replica, leading to significant overload with a variance in number of outstanding requests of 3.36×. SGL also suffers under this workload, showing high variance of 2.22× as well. Non-cache-aware policies such as LL and RR experience low cache hit rates (58.66-59.32%)

compare to SkyLB's 89.56-90.01%, and consequently deliver suboptimal throughput.

Across all experiments, the prefix tree variant (SkyLB) consistently outperforms the CH variant (SkyLB-CH) by 1.34-8.21%. This is primarily because consistent hashing can occasionally assign users with bursty request patterns to the same replica (§3.2), leading to load imbalance—since CH always routes requests to the same replica if it is available. In contrast, the prefix tree variant is more adaptive: when the prefix hit ratio is low (e.g., < 50%), it explores other underutilized replicas and distributes requests more evenly these replicas. This occasionally results in a slightly higher TTFT due to the added prefill time (e.g., in Figure 8g), but it balances the load and delivers better overall throughput (Figure 8c).

Compare to GKE Gateway, SkyLB offers key advantages through its KV Cache awareness and selective pushing mechanisms, which together contribute to 1.43-2.06× higher service throughput. In contrast, a general-purpose gateway like GKE Gateway may incur longer prefill and queuing delays due to a lack of LLM-specific optimizations.

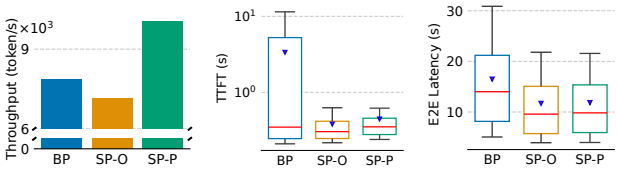
TTFT and end-to-end latency. Regarding TTFT, the P50 and mean latency are primarily affected by cross-region latency and prefill latency. SkyLB achieves the lowest P50 and mean latency, ranging from 15.87% to 57.63% of the baseline values, across all evaluated systems (Figure 8e-8h). This improvement is attributed to its geo-distributed load balancers, which reduce cross-region latency, and its high prefix hit rates, which reduce prefill time. The P90 latency, largely determined by queuing delays, can reach several seconds. Even under this constraint, SkyLB maintains the lowest P90 TTFT (10.08-23.38% of baselines), owing to its selective pushing algorithm and reduced queuing delay.

For end-to-end latency (Figures 8i-8l), SkyLB consistently delivers the best performance, achieving 1.05-2.14× improvements in P50 latency compared to baseline systems. This demonstrates that SkyLB effectively leverages KV Cache locality while maintaining balanced load.

Replica distribution. We observe that SkyLB is robust to various replica distributions, including deployments with different numbers of model replicas and varying replica ratios across regions. In our end-to-end experiments, we evaluated different configurations, such as an unbalanced distribution (3 replicas in the US, 3 in Asia, and 2 in Europe) and a balanced distribution (4 replicas per region). SkyLB consistently achieves strong performance across all scenarios.

5.2 Microbenchmark

Selective pushing by checking pending requests. We evaluate the effectiveness of the selective pushing mechanism (§3.3) using one of our baseline systems, SGLang Router. We extend the original router to incorporate two variants of selective pushing: the standard one which is based on a fixed maximum outstanding requests per replica (SP−O), and ours variant which is based on checking pending requests (SP−P). We compare both against the original version that uses blind pushing (BP). To isolate the effect of selective pushing, the experiment is conducted entirely within a single region, where all components (clients, replicas, and load balancer) are co-located. In this setup, TTFT is primarily influenced by prefill time and queuing delay.



(a) Throughput. **(b)** TTFT. **(c)** E2E latency.

Figure 9. Service Throughput and Latency, comparing Blind Pushing (BP) with two variants of Selective Pushing: fixed maximum outstanding requests per replica (SP−O) and pending request (SP−P). The TTFT latency plot is log-scaled.

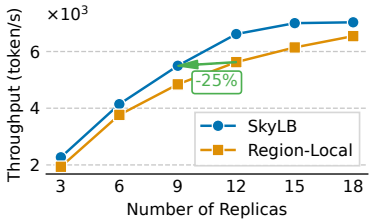


Figure 10. Service Throughput, comparing SkyLB and Region-Local deployments. We evaluate the performance using a regionally skewed workload, where the US region has 120 clients and both Asia and Europe have 40 clients. We vary the number of replicas to measure throughput gains from cross-region traffic offloading.

The experiment uses 4 replicas and 30 clients within a single region, running the Tree of Thoughts (ToT) workload with a branching factor of 2. Results are shown in Figure 9. SP−P improves service throughput by 1.27× (Figure 9a) and significantly reduces P90 TTFT by 18.47× compared to BP (Figure 9b). This demonstrates that SP−P effectively minimizes queuing delay and improves prefill time by achieving a higher KV Cache hit rate: 89.86% compared to BP's 68.89%. These gains translate into both lower latency (Figrue 9b, 9c) and higher throughput. Compared to SP−O, SP−P achieves similar TTFT but improves throughput by 1.4×, highlighting that the adaptive nature of SP−P leads to better replica utilization and overall performance under the same configuration.

Diurnal pattern. We also evaluate SkyLB under regionally imbalanced workloads to assess its performance in handling traffic patterns with diurnal pattern. We compare it against a region-local deployment strategy where each region handles requests exclusively within its own local replicas, as is common among model providers (Figure 1(a)). Specifically, we

simulate a regionally skewed workload scenario representative of typical US working hours, where the US region uses 120 clients, while both Asia and Europe have 40 clients. We vary the total number of replicas deployed in both SkyLB and the region-local baseline, with replicas are evenly distributed across the three regions.

The throughput results for both systems are shown in Figure 10. With an equal number of replicas, SkyLB consistently outperforms the region-local system by between 1.07× and 1.18×, demonstrating the effectiveness of cross-region traffic handling to offload traffic onto regions with less load. Moreover, we observe that SkyLB achieves comparable throughput with only 9 replicas as the region-local deployment achieves with 12 replicas, translating into a cost reduction of 25% while maintaining the same level of throughput.

6 Related Work

Load balancing for CPU workloads. Efficiently managing CPU resources for latency-sensitive applications is a well-studied problem in CPU workloads. Prior work proposes load balancing policies that make task distribution decisions at microsecond-scale latencies. McClure et al. [35] classify these techniques into two categories: *work stealing* [10, 19, 21, 30, 33, 41, 43] and *work shedding* [39]. In work stealing, idle CPU cores actively pull jobs from overloaded cores. In contrast, work shedding involves overloaded cores pushing excess jobs to other cores. Empirical studies show that work stealing generally outperforms work shedding in terms of both latency and CPU utilization.

Production systems. Amazon Bedrock [5] is a fully managed LLM inference service that supports cross-region inference to handle traffic spikes. However, its offloading is limited to within the same continent, missing the opportunity to aggregate diurnal patterns. Additionally, Bedrock is a hosted solution operating at AWS scale, whereas SkyLB is a self-hosted serving system designed for broader accessibility. GCP Gateway [24] provides a unified endpoint for global deployment by routing requests across multiple GKE [2] clusters. However, this solution is not tailored for LLM workloads. Neither Bedrock nor GCP Gateway incorporates prefix awareness, thereby failing to reuse KV Cache and reduce compute overhead. They also lack selective pushing based on pending requests, making them more susceptible to replica overload.

Prefix-aware load balancing. Prior work has explored leveraging KV Cache reuse to improve the efficiency of LLM request routing. Preble [51] achieves low latency and high throughput by maximizing prefix cache hit rates, but it relies on a centralized global scheduler, limiting its applicability to a single-region setting. Similarly, SGLang Router [1] maintains a global prefix tree across all replicas, incorporating more fine-grained load balancing policies. DLPM [12] introduces a scheduling algorithm that improves upon Preble in both latency and throughput while also offering fairness guarantees to clients. While these centralized approaches deliver high performance, their reliance on a single scheduler makes them unsuitable for cross-region, production-grade deployments due to high inter-region communication latency and the inherent risk of a single point of failure.

Improving GPU utilization through job colocation. Many techniques has been proposed to improve GPU utilization by sharing resources either spatially or temporally [13, 18, 37, 42, 58, 59, 63, 66]. These include strategies such as colocating training and serving jobs or enabling multi-model serving on shared GPUs. However, due to the strict service-level objectives (SLOs) associated with serving tasks, job interference remains a concern that can degrade performance. Moreover, in many real-world settings, users may only run serving workloads, limiting the opportunities for job colocation.

7 Future Work

Support for heterogeneous accelerators. Load balancing is more challenging in heterogeneous environments. While SkyLB currently focuses on homogeneous replicas, it can be extended to support heterogeneous accelerators, such as different GPU types or other hardware like TPUs [28] and AWS Inferentia [4]. Notably, the selective pushing by checking pending requests mechanism in SkyLB is inherently hardware-agnostic: it identifies overloaded replicas without relying on hardware-specific features, making it naturally compatible with heterogeneous settings. However, the prefix-aware routing and overall load balancing policies remain as an open question.

More advanced policies. Request characteristics, such as prompt length, can influence ideal routing strategies. For instance, shorter prompts incur lower prefill costs, making it more advantageous to route them to replicas with slightly lower load instead of prioritize prefix reuse. SkyLB can be extended to incorporate request-characteristic aware routing strategies, dynamically adapting its decision-making process based on each request.

8 Conclusion

Shifting regional diurnal patterns make cost-efficient LLM deployment in multi-region setups challenging. To solve this problem, we propose SkyLB, a locality-aware cross-region load balancer for LLM inference. Through its two-layer design and regional snapshots, SkyLB enables cross-region coordination on prefix locality among geo-distributed load balancers. By checking pending requests and performing selective pushing before routing, SkyLB avoids overloaded replicas and maintains a balanced load across all replicas. Together, these techniques improve GPU utilization and reduce serving costs while maintaining low latency and high throughput. Our extensive evaluation across real-world and synthetic

scenarios show that SkyLB consistently outperforms existing production and research systems, achieving 1.12-2.06× higher throughput and 1.74-6.30× lower latency, and 25% cost savings compare to other systems.

References

[1] 2024. SGLang v0.4: Zero-Overhead Batch Scheduler, Cache-Aware Load Balancer, Faster Structured Outputs. https://lmsys.org/blog/2024-12-04-sglang-v0-4/. Accessed: 2025-05-12.

[2] 2025. Google Kubernetes Engine (GKE). https://cloud.google.com/kubernetes-engine?hl=en. Accessed: 2025-05-12.

[3] AIME. 2025. *CLOUD VS. ON-PREMISE - Total Cost of Ownership Analysis*. https://www.aime.info/blog/en/cloud-vs-on-premise-total-cost-of-ownership-analysis/ Accessed: 2025-05-13.

[4] Amazon Web Service. 2025. *AWS Inferentia*. https://aws.amazon.com/ai/machine-learning/inferentia/ Accessed: 2025-05-15.

[5] Amazon Web Services. 2025. Amazon Bedrock. https://aws.amazon.com/bedrock/ Accessed: 2025-05-09.

[6] Amazon Web Services. 2025. *Amazon Route 53*. https://aws.amazon.com/route53/ Accessed: 2025-05-12.

[7] Amazon Web Services. 2025. *Cloud Computing with AWS — Build, Deploy, and Manage Websites, Apps or Processes on AWS Secure, Reliable Network.* https://aws.amazon.com/ Accessed: 2025-05-13.

[8] Apple Inc. 2025. *Use ChatGPT with Apple Intelligence on iPhone*. https://support.apple.com/guide/iphone/use-chatgpt-with-apple-intelligence-iph00fd3c8c2/ios Accessed: 2025-05-12.

[9] Mohammad Beigi, Sijia Wang, Ying Shen, Zihao Lin, Adithya Kulkarni, Jianfeng He, Feng Chen, Ming Jin, Jin-Hee Cho, Dawei Zhou, Chang-Tien Lu, and Lifu Huang. 2024. Rethinking the Uncertainty: A Critical Review and Analysis in the Era of Large Language Models. arXiv:2410.20199 [cs.AI] https://arxiv.org/abs/2410.20199

[10] Edouard Bugnion. 2017. ZygOS: Achieving Low Tail Latency for Microsecond-scale Networked Tasks. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*.

[11] Brian Buntz. 2024. *Musk unveils world's largest AI cluster, OpenAI eyes premium subscriptions*. https://www.rdworldonline.com/this-week-in-ai-musk-unveils-worlds-largest-ai-cluster-openai-eyes-2000-month-subscriptions/ Accessed: 2025-05-13.

[12] Shiyi Cao, Yichuan Wang, Ziming Mao, Pin-Lun Hsu, Liangsheng Yin, Tian Xia, Dacheng Li, Shu Liu, Yineng Zhang, Yang Zhou, et al. 2025. Locality-aware Fair Scheduling in LLM Serving. *arXiv preprint arXiv:2501.14312* (2025).

[13] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2021. Multi-model machine learning inference serving with gpu spatial partitioning. *arXiv preprint arXiv:2109.01611* (2021).

[14] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[15] Consensus. 2025. Consensus: AI-Powered Academic Search Engine. https://consensus.app/ Accessed: 2025-05-12.

[16] Cursor. 2025. *Cursor: The AI Code Editor*. https://www.cursor.com/en Accessed: 2025-05-15.

[17] Mark Van der Boor, Sem C. Borst, Johan S. H. Van Leeuwaarden, and Debankur Mukherjee. 2022. Scalable Load Balancing in Networked Systems: A Survey of Recent Advances. *SIAM Rev.* 64, 3 (Aug. 2022), 554–622. doi:10.1137/20m1323746

[18] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. 2020. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 492–506.

[19] James Dinan, D Brian Larkins, Ponnuswamy Sadayappan, Sriram Krishnamoorthy, and Jarek Nieplocha. 2009. Scalable work stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 1–11.

[20] Envoy Project. 2025. *Load Balancers — Envoy Proxy Documentation*. https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/load_balancers Accessed: 2025-05-13.

[21] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating interference at microsecond timescales. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 281–297.

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

[23] GitHub. 2025. *GitHub Copilot: Your AI pair programmer*. https://github.com/features/copilot Accessed: 2025-05-12.

[24] Google Cloud. 2025. *About the Gateway API | GKE networking*. https://cloud.google.com/kubernetes-engine/docs/concepts/gateway-api Accessed: 2025-05-12.

[25] Google Cloud. 2025. *Compute Engine Reservations Overview*. https://cloud.google.com/compute/docs/instances/reservations-overview Accessed: 2025-05-12.

[26] Google Cloud Platform. 2025. *Google Cloud Platform — Future-proof infrastructure. Powerful data and analytics. No ops, just code.* https://cloud.google.com/ Accessed: 2025-05-13.

[27] Google LLC. 2025. *Introducing Gemini: Your New Personal AI Assistant*. https://gemini.google/assistant/ Accessed: 2025-05-12.

[28] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR] https://arxiv.org/abs/2304.01433

[29] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. 1997. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, USA) *(STOC '97)*. Association for Computing Machinery, New York, NY, USA, 654–663. doi:10.1145/258533.258660

[30] Martin Karsten and Saman Barghi. 2020. User-level threading: Have your cake and eat it too. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 1 (2020), 1–30.

[31] Kubernetes Authors. 2025. *Services, Load Balancing, and Networking*. https://kubernetes.io/docs/concepts/services-networking/ Accessed: 2025-05-13.

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[33] Jing Li, Kunal Agrawal, Sameh Elnikety, Yuxiong He, I-Ting Angelina Lee, Chenyang Lu, and Kathryn S McKinley. 2016. Work stealing for interactive services to meet target latency. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1–13.

[34] Ziming Mao, Tian Xia, Zhanghao Wu, Wei-Lin Chiang, Tyler Griggs, Romil Bhardwaj, Zongheng Yang, Scott Shenker, and Ion Stoica. 2025. SkyServe: Serving AI Models across Regions and Clouds with Spot Instances. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 159–175.

doi:10.1145/3689031.3717459

[35] Sarah McClure, Amy Ousterhout, Scott Shenker, and Sylvia Ratnasamy. 2022. Efficient scheduling policies for Microsecond-Scale tasks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1–18.

[36] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. Spotserve: Serving generative large language models on preemptible instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1112–1127.

[37] Jaiaid Mobin, Avinash Maurya, and M Mustafa Rafique. 2023. Colti: Towards concurrent and co-located dnn training and inference. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*. 309–310.

[38] Evan Morikawa. 2023. Behind the Scenes Scaling ChatGPT. https://youtu.be/PeKMEXUrlq4?t=833. Accessed: 2025-05-25.

[39] Malarvizhi Nandagopal, Kandaswamy Gokulnath, and V Rhymend Uthariaraj. 2010. Sender initiated decentralized dynamic load balancing for multi cluster computational grid environment. In *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*. 1–4.

[40] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. 2018. Vesper: Measuring Time-to-Interactivity for Web Pages. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 217–231. https://www.usenix.org/conference/nsdi18/presentation/netravali-vesper

[41] Poornima Nookala, Kyle Chard, and Ioan Raicu. 2024. X-OpenMP—eXtreme fine-grained tasking using lock-less work stealing. *Future Generation Computer Systems* 159 (2024), 444–458.

[42] Gabriele Oliaro, Xupeng Miao, Xinhao Cheng, Vineeth Kada, Ruohan Gao, Yingyi Huang, Remi Delacourt, April Yang, Yingcheng Wang, Mengdi Wu, et al. 2024. Flexllm: A system for co-serving large language model inference and parameter-efficient finetuning. *arXiv preprint arXiv:2402.18789* (2024).

[43] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 361–378.

[44] Archit Patke, Dhemath Reddy, Saurabh Jha, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2025. Hierarchical Autoscaling for Large Language Model Serving with Chiron. *arXiv preprint arXiv:2501.08090* (2025).

[45] Perplexity AI. 2025. *Perplexity: AI-powered answer engine that provides accurate, trusted, and real-time answers to any question.* https://www.perplexity.ai Accessed: 2025-05-15.

[46] PYMNTS. 2025. *Sam Altman: OpenAI Has Reached Roughly 800 Million Users*. https://www.pymnts.com/artificial-intelligence-2/2025/sam-altman-openai-has-reached-roughly-800-million-users/ Accessed: 2025-05-13.

[47] Ray Project. 2025. *Ray Serve: Scalable and Programmable Serving for ML Models*. https://docs.ray.io/en/latest/serve/index.html Accessed: 2025-05-13.

[48] Replit Inc. 2025. Intro to Ghostwriter. https://replit.com/learn/intro-to-ghostwriter Accessed: 2025-05-12.

[49] Sherwood News. 2024. *Just four companies are hoarding tens of billions of dollars worth of Nvidia GPU chips*. https://sherwood.news/tech/companies-hoarding-nvidia-gpu-chips-meta-tesla/ Accessed: 2025-05-14.

[50] Anton Shilov. 2024. *TikTok owner ByteDance taps TSMC to make its own AI GPUs to stop relying on Nvidia*. https://www.tomshardware.com/tech-industry/artificial-intelligence/tiktok-owner-bytedance-taps-tsmc-to-make-its-own-ai-gpus-to-stop-relying-on-nvidia-the-company-has-reportedly-spent-over-dollar2-billion-on-nvidia-ai-gpus Accessed: 2025-05-12.

[51] Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. 2024. Preble: Efficient Distributed Prompt Scheduling for LLM Serving. (2024). arXiv:2407.00023 [cs.DC] https://arxiv.org/abs/2407.00023

[52] Steve Roberts. 2023. *Amazon CodeWhisperer, Free for Individual Use, is Now Generally Available*. https://aws.amazon.com/blogs/aws/amazon-codewhisperer-free-for-individual-use-is-now-generally-available/ Accessed: 2025-05-12.

[53] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on networking* 11, 1 (2003), 17–32.

[54] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2024. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. arXiv:2408.00741 [cs.AI] https://arxiv.org/abs/2408.00741

[55] Tabnine Inc. 2025. *Tabnine: AI Code Assistant*. https://www.tabnine.com/ Accessed: 2025-05-12.

[56] Windsurf. 2025. *Windsurf: The Most Powerful AI Code Editor*. https://windsurf.com/ Accessed: 2025-05-12.

[57] Yuxing Xiang, Xue Li, Kun Qian, Wenyuan Yu, Ennan Zhai, and Xin Jin. 2025. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. arXiv:2505.09999 [cs.DC] https://arxiv.org/abs/2505.09999

[58] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 595–610.

[59] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 533–548.

[60] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, et al. 2023. SkyPilot: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 437–455.

[61] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL] https://arxiv.org/abs/2305.10601

[62] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[63] Shan Yu, Jiarong Xing, Yifan Qiao, Mingyuan Ma, Yangmin Li, Yang Wang, Shuo Yang, Zhiqiang Xie, Shiyi Cao, Ke Bao, et al. 2025. Prism: Unleashing GPU Sharing for Cost-Efficient Multi-LLM Serving. *arXiv preprint arXiv:2505.04021* (2025).

[64] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting cloud services for Cost-Effective,SLO-Aware machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1049–1062.

[65] Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. WildChat: 1M ChatGPT Interaction Logs in the Wild. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=Bl8u7ZRlbM

[66] Yihao Zhao, Xin Liu, Shufan Liu, Xiang Li, Yibo Zhu, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Muxflow: Efficient and safe gpu

sharing in large-scale production deep learning clusters. *arXiv preprint arXiv:2303.13803* (2023).

[67] 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. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL]

[68] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems* 37 (2024), 62557–62583.

[69] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response length perception and sequence scheduling: An llm-empowered llm inference pipeline. *Advances in Neural Information Processing Systems* 36 (2023), 65517–65530.