SplitScaling: Adaptive Scaling for Disaggregated LLM Serving Against Traffic Bursts via DRL

Wei Xiao, **Xuefeng Huang**, **Weijia Shi** and **Baokang Zhao**
National University of Defense Technology
xw@nudt.edu.cn, hxfhuangxuefeng@foxmail.com, swj957@nudt.edu.cn, bkzhao@nudt.edu.cn

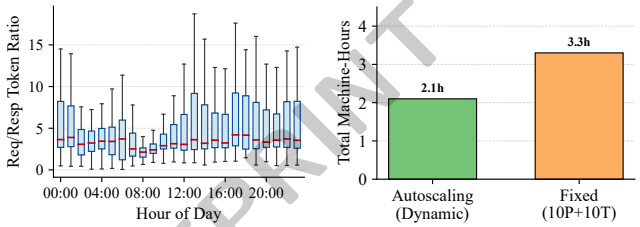
Abstract

The disaggregated Prefill-Decode (PD) architecture has emerged as a prominent paradigm for efficient Large Language Model inference serving. However, resource management remains a critical challenge, particularly under the dual burstiness of real-world scenarios—characterized by volatile fluctuations in both request arrival rates and Prompt-to-Response ratios. Existing rule-based heuristics often fail to accurately identify system bottlenecks, leading to severe resource misallocation and Service Level Objective (SLO) violations. To address this, we propose a Deep Reinforcement Learning-based auto-scaling framework tailored for the PD architecture. By modeling the resource allocation problem as a Markov Decision Process, our framework enables the agent to capture non-linear load dynamics, thereby achieving decoupled and precise scaling for prefill and decode pools. Furthermore, to mitigate Head-of-Line blocking caused by scaling latency, we design an immediate rescheduling mechanism that migrates queued tasks to newly ready nodes in real-time. Experimental results driven by Azure bursty load traces demonstrate that our framework significantly reduces computational costs by 25.2% and 28.3% compared to the Static configuration and HeteroScale, respectively, while strictly adhering to SLOs.

1 Introduction

The escalating scale of Large Language Models (LLMs) makes inference costs a primary bottleneck for deployment [Kwon *et al.*, 2023; Achiam *et al.*, 2023; Touvron *et al.*, 2023]. Generative inference comprises two distinct phases: the compute-bound Prefill and memory-bound Decode [Dao *et al.*, 2022]. Traditional coupled architectures suffer from resource interference between these phases, degrading throughput [Zhong *et al.*, 2024]. To address this, disaggregated Prefill-Decode (PD) architectures like Splitwise [Patel *et al.*, 2024] and Mooncake [Qin *et al.*, 2025] have emerged, optimizing resource utilization through hardware heterogeneity.

In real-world production environments, LLM services encounter challenging workloads characterized by dual burstiness. As illustrated in Fig. 1a (based on real-world traces [Wang *et al.*, 2025]), this traffic pattern exhibits not only transient surges in Request Per Second (RPS) but also drastic fluctuations in request structure—specifically, the ratio between prompt input length and token generation length.



(a) Distribution of request-to-response token ratios.

(b) Total machine-hours comparison.

Figure 1: Analysis of workload characteristics and system resource efficiency.

Such load heterogeneity poses a severe challenge to resource provisioning: static resource partitioning (i.e., fixed PD instance ratios) cannot simultaneously accommodate compute-bound prompt bursts and memory-bound generation bursts. Our experiments in Fig. 1b quantify the consequences of this mismatch: Ideally, under bursty load scenarios, a dynamic auto-scaling mechanism can reduce infrastructure costs by 36.3% compared to static configuration schemes by reclaiming idle resources on demand, while strictly adhering to End-to-End Service Level Objectives (SLOs). This indicates that traditional static configurations are rendered ineffective in large-scale bursty scenarios, necessitating an elastic scaling framework capable of perceiving load characteristics and dynamically adjusting heterogeneous resource ratios.

Despite the criticality of dynamic resource management, existing solutions fall short when handling highly dynamic bursty traffic. First, traditional Kubernetes Horizontal Pod Autoscaler (HPA) mechanisms [Rzadca *et al.*, 2020] rely solely on CPU or memory utilization. They lack awareness of the heterogeneous computational characteristics of the complex internal phases of LLM inference (Prefill and Decode), leading to a misalignment beten scaling decisions and actual load demands. Second, although advanced works like HeteroScale [Li *et al.*, 2025] attempt to optimize PD instance

ratios, they typically employ rigid strategies based on rules or exhaustive search. Such coupled scaling often results in instance imbalance—for example, when the Prefill phase becomes a bottleneck, rigid ratio adjustments may leave Decode resources idle (or vice versa), ultimately causing system throughput to collapse due to front-end blocking.

Furthermore, existing schemes generally overlook the scaling lag: even after new instances are launched, long-tail requests accumulated in the overloaded queues of old instances must still wait, rendering the scaling operation too late to avert impending SLO violations [San Juan and Wong, 2023; Fu *et al.*, 2024].

To address these challenges, we propose an elastic scaling framework evolved from the Splitwise architecture. This framework aims to break the limitations of traditional rigid strategies by decoupling decision-making from execution to achieve precise responses to heterogeneous loads. To handle complex traffic fluctuations, we formulate the dynamic resource provisioning problem as a Markov Decision Process (MDP) and introduce a Deep Reinforcement Learning (DRL) agent [Haarnoja *et al.*, 2018; Xue *et al.*, 2022; Fan *et al.*, 2022; Ma *et al.*, 2024; Mao *et al.*, 2016; Mao *et al.*, 2017; Mao *et al.*, 2019]. Unlike rule-based heuristics, this agent perceives fine-grained resource pressure differences between Prefill and Decode phases, adaptively learning the non-linear mapping between load characteristics and resource ratios. Moreover, addressing the queuing issues caused by scaling lag, we design a Just-in-Time Rescheduling mechanism. This allows backlogged tasks to be instantly migrated to newly scaled or just-released nodes, strictly guaranteeing SLO constraints under bursty loads.

Our contributions are summarized below:

- **DRL-Driven Disaggregated Auto-Scaling:** We present the first application of Reinforcement Learning to solve the dynamic resource balancing problem in the PD disaggregated architecture for LLM services. Leveraging the strong state-space exploration capabilities of DRL, we dynamically learn the non-linear relationship between system load structure and resource pools, achieving independent and precise scaling for Prefill and Decode resources to effectively resolve resource imbalance under heterogeneous loads.
- **Lag-Eliminating Rescheduling Strategy:** We propose and implement a highly efficient task rescheduling strategy. By dynamically migrating tasks from overloaded queues to new instances, we eliminate the Head-of-Line (HOL) blocking effect inherent in traditional scaling schemes, significantly reducing tail latency in bursty scenarios and enabling new resources to effectively avert imminent SLO violations.
- **Bursty Workload Benchmarking and Verification:** We construct an experimental testbed based on Splitwise capable of high-fidelity simulation of production bursty traffic. Experiments demonstrate that compared to the Static configuration and HeteroScale, our proposed scheme reduces queue waiting time by 17.2% and 85.2%, respectively, while achieving 25.2% and 28.3% cost optimization under the same SLO constraints.

2 Related Work

2.1 LLM Serving Systems

Efficient serving of LLMs faces the dual challenges of massive memory footprint and autoregressive generation characteristics. Pioneering work like Orca [Yu *et al.*, 2022] introduced iteration-level scheduling, effectively reducing pipeline bubbles caused by variable-length requests and significantly improving throughput compared to static batching. vLLM [Kwon *et al.*, 2023] addressed memory fragmentation by proposing the PagedAttention mechanism, which utilizes non-contiguous KV Cache management inspired by virtual memory to substantially enhance memory utilization. Additionally, AlpaServe [Li *et al.*, 2023] optimized job completion times through model parallelism optimization.

While these systems [Yu *et al.*, 2022; Kwon *et al.*, 2023; Li *et al.*, 2023; Sheng *et al.*, 2023; Zheng *et al.*, 2024; Miao *et al.*, 2024a; Li *et al.*, 2024; Agrawal *et al.*, 2024; Wu *et al.*, 2023; Aminabadi *et al.*, 2022] greatly optimize execution efficiency within a single instance, they typically assume a fixed resource budget for the cluster. They lack the capability to respond to bursty traffic scenarios [Wang *et al.*, 2025] at the cluster level and cannot dynamically adjust the number of instances. Consequently, when ingress traffic instantaneously exceeds the static capacity of a single node, severe queuing delays ensue.

2.2 Disaggregated Prefill-Decode Architectures

To resolve resource contention and interference between the compute-bound Prefill phase and the memory-bound Decode phase, recent research has shifted towards Disaggregated PD architectures [Zhong *et al.*, 2024; Patel *et al.*, 2024; Qin *et al.*, 2025; Hu *et al.*, 2025; Zhang *et al.*, 2025]. DistServe [Zhong *et al.*, 2024] systematically analyzed the interference patterns between these phases and proposed decoupling them into distinct GPU clusters, demonstrating significant advantages in goodput. Similarly, Splitwise [Patel *et al.*, 2024] separates prompt processing from token generation to optimize hardware utilization, while Mooncake [Qin *et al.*, 2025] proposes a KVCache-centric architecture to optimize high-bandwidth context transfer between disaggregated pools [Miao *et al.*, 2024b].

Although these works establish the benefits of disaggregated architectures, they largely treat the ratio of Prefill to Decode instances as a static configuration or rely on offline analysis. In real-world production environments, the ratio of prompt length to generation length fluctuates drastically. A static PD ratio leads to severe structural resource mismatch, such as Decode instances idling while waiting for saturated Prefill instances. Our work builds upon the PD disaggregated architecture but introduces a real-time dynamic resource adjustment mechanism in the control plane.

2.3 Autoscaling and Dynamic Resource Management

Heuristic-based Autoscaling: The industry-standard Kubernetes HPA [Rzadca *et al.*, 2020] relies on simple metrics like CPU or memory utilization for scaling. However, its inability to perceive LLM-specific bottlenecks leads to a disconnection

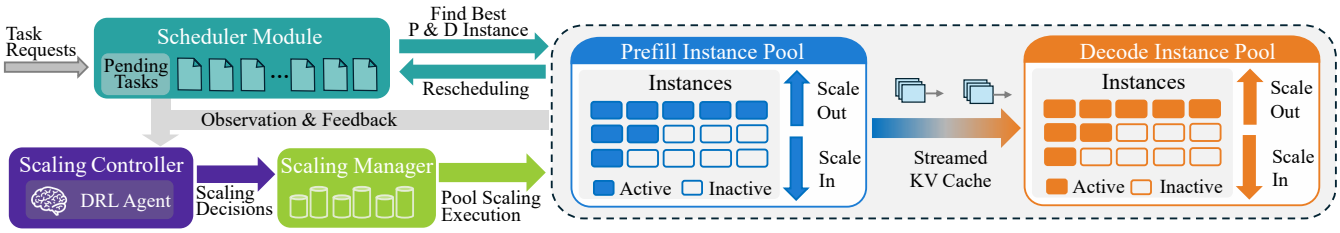


Figure 2: Overview of the proposed adaptive auto-scaling framework.

between scaling decisions and actual load demands. A typical example is the KV Cache Blocking phenomenon: when a large number of requests are in the Decode phase, GPU memory may be exhausted by long sequences, forcing new requests to queue. Yet, since the Decode phase is memory-bound, GPU compute utilization often remains low. This illusion of low utilization but saturated service deceives HPA into remaining silent, triggering severe queuing backlogs and SLO violations. More advanced heuristic methods like HeteroScale [Li *et al.*, 2025] attempt to search for the optimal PD instance ratio, but these approaches are typically based on rigid rules that sever the dependency between Prefill and Decode phases, making them prone to local optima and resource oscillation.

To fill this gap, we propose a unified single-agent DRL framework that achieves collaborative scaling by capturing the non-linear dependencies between Prefill and Decode. Uniquely, we introduce a Rescheduling mechanism that proactively eliminates HOL blocking during the scaling transition period, ensuring strict adherence to SLOs even under extreme bursty scenario s.

3 Overview of the Adaptive Scaling Framework

To address the aforementioned challenges of high load dynamics and structural heterogeneity, we propose an adaptive auto-scaling framework tailored for disaggregated inference architectures. Specifically, the framework aims to overcome the resource imbalance and scaling lag caused by traditional strategies by decoupling resource decision-making from task scheduling. As shown in Fig. 2, the core of the system comprises three tightly coupled components: the Scheduler Module, the Scaling Controller, and the Scaling Manager.

In the overall workflow, the Scheduler Module acts as the traffic entry point, smoothing transient burst requests via a global queue mechanism while providing a real-time load observation window for upper-layer decision-making. The Scaling Controller serves as the system's brain; it utilizes a DRL agent to extract fine-grained state features from the scheduler and instance pools. Under the premise of satisfying SLO constraints, it formulates optimal scaling strategies targeting the two orthogonal action spaces of Prefill and Decode. Subsequently, Scaling Decisions are dispatched to the Scaling Manager, which governs the lifecycle of instance resources, executing precise scale-out and scale-in operations. The design details of each key component are elaborated below.

3.1 Scheduler Module

To maximize system throughput, existing solutions [Patel *et al.*, 2024] typically employ a streaming KV Cache transmission mechanism that overlaps computation with communication. This mechanism utilizes layer-wise asynchronous transmission of KV Cache, allowing cross-node network communication of the current layer's data to proceed in parallel with the forward computation of subsequent layers. This effectively hides data transmission latency within the pipeline execution of the Prompt phase. While this compute-transfer overlapping pipeline mode reduces end-to-end latency, it introduces stringent scheduling constraints: Prefill and Decode instances must achieve Joint Allocation within a specific time window.

However, in resource-constrained scenarios, this coupling creates significant performance bottlenecks. Decode instance memory is often under high load, and its release exhibits latency. Forcing the scheduling of a Prefill task under these conditions results in generated KV Cache being unable to transfer due to a lack of available memory at the receiving end (Decode), thereby triggering HOL blocking. This transmission stagnation forces the Prefill instance into a suspended wait state, preventing its expensive computational resources from being effectively released or reused. Splitwise circumvents this by falling back Prefill and Decode tasks to the same instance, but this sacrifices the core advantages of PD disaggregation, inducing severe computational interference and degraded mixed-batch efficiency.

To this end, we design an Atomic Scheduler based on a global queue. When making decisions, the scheduler simultaneously checks resource availability on both the Prefill and Decode sides. If either link lacks suitable instances, the task is not scheduled and remains suspended in the global scheduling queue.

This deferred binding strategy provides a critical time window for Re-scheduling. Once the system triggers the scaling mechanism, newly launched instances immediately register with the scheduler. Similarly, when any active instance completes an inference task and releases memory, the scheduler instantly perceives this availability. Since tasks remain resident in the global queue, the scheduler can instantaneously redirect accumulated tasks to these newly ready nodes (whether newly scaled or just released), rather than being deadlocked in the local queues of overloaded old instances. This mechanism dynamically achieves load balancing and effectively eliminates instance-level queuing backlogs, thereby strictly guaranteeing SLO constraints under vi-

olently fluctuating loads.

3.2 Scaling Controller

The Scaling Controller provides an algorithm-agnostic interface capable of flexibly supporting decision algorithms ranging from simple heuristic rules to complex machine learning models, such as PID control or linear programming. Unlike unified scaling strategies in traditional monolithic architectures, our framework fully leverages the characteristics of PD disaggregation by modeling the Prefill instance pool and Decode instance pool as two orthogonal action spaces.

This module collects multi-dimensional system states at fixed intervals—including request arrival rates, global queue lengths, and resource utilization rates of each instance pool. To capture the non-linear mapping of performance bottlenecks under complex traffic patterns, we instantiate a global controller based on DRL. The intelligent agent can make precise judgments based on observations: when abnormal latency is detected in the Prefill phase, it specifically expands Prefill instances to accelerate time-to-first-token; if KV Cache backlog or limited Decode throughput occurs, it prioritizes increasing Decode instances. This fine-grained decision mechanism effectively avoids blind resource over-provisioning. (Specific DRL design details will be discussed in Section 4.)

3.3 Scaling Manager

Upon receiving decision signals, this module is responsible for the rapid provisioning and reclamation of virtualized resources.

During the resource scale-out phase, GPU instances incur startup latency due to virtual machine booting, container initialization, and large-scale model weight loading. New instances register with the Atomic Scheduler only after startup is complete. At the precise moment resources become formally available, the scheduler directs backlogged tasks from the global queue to the new instances, thereby maximizing the utilization efficiency of new resources.

During the resource scale-in phase, to prevent service interruptions caused by abrupt instance termination, we introduce a strict Connection Draining mechanism. Once an instance is marked for removal by the decision module, the manager immediately notifies the scheduling layer to place it into a drain-only silent state (accepting no new requests). The instance continues to process existing requests in memory until all KV Cache is fully released and the current inference task concludes normally. Through this Graceful Decommissioning strategy, the framework ensures that the system maintains high reliability and SLO achievement rates even during drastic dynamic adjustments in cluster size.

4 Deep Reinforcement Learning Design

To address the complex non-linear coupling between load volatility and heterogeneous resource constraints, the Scaling Controller incorporates a decision-making agent based on DRL, as illustrated in Fig. 3. Unlike static heuristic rules, this module perceives multi-dimensional system states in real-time, adaptively learning the mapping between traffic characteristics and performance bottlenecks. Consequently, it generates precise scaling strategies within two orthogonal action

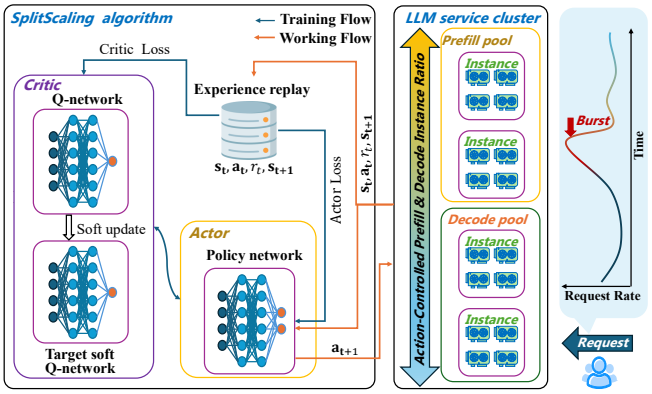


Figure 3: Architecture of the DRL-based Auto-scaling Controller. We leverage the Soft Actor-Critic (SAC) algorithm to learn optimal scaling policies for the PD disaggregated cluster.

spaces—Prefill and Decode—thereby maximizing resource efficiency while strictly guaranteeing SLOs.

We formulate the auto-scaling problem for PD disaggregated LLM serving as a MDP, defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. This formulation is designed to resolve the differentiated resource demands of the Prefill and Decode phases, as well as the system coupling constraints introduced by streaming KV Cache transmission.

4.1 State Space ($\mathcal{S}$)

To enable the RL agent to capture the complex dynamics of the LLM inference system and make optimal decisions, we construct a multi-dimensional feature vector as the state space $\mathcal{S}$. The design of state $s_t \in \mathcal{S}$ follows MDP principles, aiming to comprehensively characterize system load, resource utilization, and workload attributes. We categorize the state space into four key dimensions:

1) Traffic Patterns: Given the non-stationarity of inference request arrivals, the state includes traffic evolution trends:

- **Request Rate (λ_t):** The current Requests Per Second (RPS), reflecting the macroscopic load on the system.
- **Traffic Dynamics ($\Delta\lambda_t$):** Calculated as $\lambda_t - \lambda_{t-1}$. This feature represents the trend and dynamics of traffic changes, allowing the agent to perceive traffic bursts and make preemptive strategy adjustments.
- **Temporal Info (τ_t):** Provides cyclic temporal features (normalized timestamp) to capture periodic patterns or cycles in the workload.

2) System Pressure and Bottlenecks: These features directly reflect the backlog level within the scheduler and resource saturation:

- **Scheduler Queue (Q_{len}):** The total number of pending prompts and decode tokens in the scheduler. This is the core signal for assessing the risk of SLO violations.
- **Rejection Status (C_{none}):** Records the frequency of allocation failures due to insufficient resources. This sparse signal is critical for identifying memory fragmentation or computational bottlenecks.

3) Resource Allocation Status: Describes the current distribution and internal state of computational resources:

- **Resource Scale (n_p, n_t):** The number of active instances allocated to Prompt and Token phases, respectively, allowing the agent to perceive the current scale of provisioned resources.
- **Instance Backlog ($B_{pending}$):** Specifically refers to the normalized number of tokens pending in the internal queues of Prompt instances. Unlike the global queue, this feature provides fine-grained visibility into the execution pipeline depth at the instance layer.

4) Workload Characteristics: Since LLM inference latency is highly sensitive to sequence length, we extract the following features:

- **Token Length (L_{avg}):** The average input or output length of requests in the queue, which determines the expected request complexity and future computational intensity.
- **Arrival Intensity (R_{gen}):** Describes the intensity or density of arriving tasks (tokens) per unit time. Through the relationship between Q_{len} and this intensity metric, the agent can implicitly infer system congestion levels.

Finally, all continuous features are normalized or standardized to eliminate scaling effects and enhance the numerical stability of training. This state representation provides the policy network with a quasi-Markovian observation environment, enabling it to distinguish between transient noise and structural changes in system demand.

4.2 Action Space ($\mathcal{A}$)

Unlike unified scaling in monolithic architectures, the PD disaggregated architecture allows for the decoupled provisioning of compute and memory resources. To accommodate the SAC algorithm, we define a continuous action space $\mathcal{A}$ as follows:

$$\mathbf{a}_t = [\alpha_{p,t}, \alpha_{d,t}]^\top, \quad \alpha_{p,t}, \alpha_{d,t} \in [-1, 1] \tag{1}$$

Here, $\alpha_{p,t}$ and $\alpha_{d,t}$ correspond to the scaling magnitude values for the Prompt instance pool and Token instance pool, respectively. In the execution phase, these continuous values are mapped to concrete discrete operations (e.g., adding or removing k instances). This decoupled action design empowers the agent to independently address computational bottlenecks (by scaling Prompt instances) or memory/bandwidth bottlenecks (by scaling Token instances).

4.3 Reward Function (R)

To achieve an optimal balance between Quality of Service (QoS) and resource efficiency, we construct a multi-objective reward function. This function aims to maximize SLO satisfaction while minimizing computational costs via a penalty mechanism:

$$R = w_s \cdot \left(\sum_{i \in \mathcal{K}} (r_{ttft}^{(i)} + r_{tbt}^{(i)}) + 1_{all} \cdot \mathcal{B} \right) - w_c \cdot (n_p + n_t) \tag{2}$$

where w_s and w_c are weight coefficients, and n_p, n_t represent the number of active Prompt and Token instances. For each latency tier $i \in \mathcal{K}$, the reward terms $r_{ttft}^{(i)}$ and $r_{tbt}^{(i)}$ are defined using a piecewise function: if the latency v meets the SLO, a fixed utility is awarded; if violated, a linear penalty $\max(0, \alpha - \beta \frac{v - SLO}{SLO})$ is introduced, where the decay rate is controlled by the gradient factor β. The indicator function 1_{all} triggers a global bonus $\mathcal{B}$ only when all SLOs are met, thereby reinforcing the system's global stability under multiple constraints.

4.4 Optimization Objective

The goal of the RL agent is to learn the optimal policy π^* that maximizes the expected cumulative discounted return:

$$J(\pi) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \right] \tag{3}$$

where $\gamma \in [0, 1)$ is the discount factor, regulating the agent's focus between immediate rewards and long-term returns.

5 Evaluation

5.1 Experimental Setup

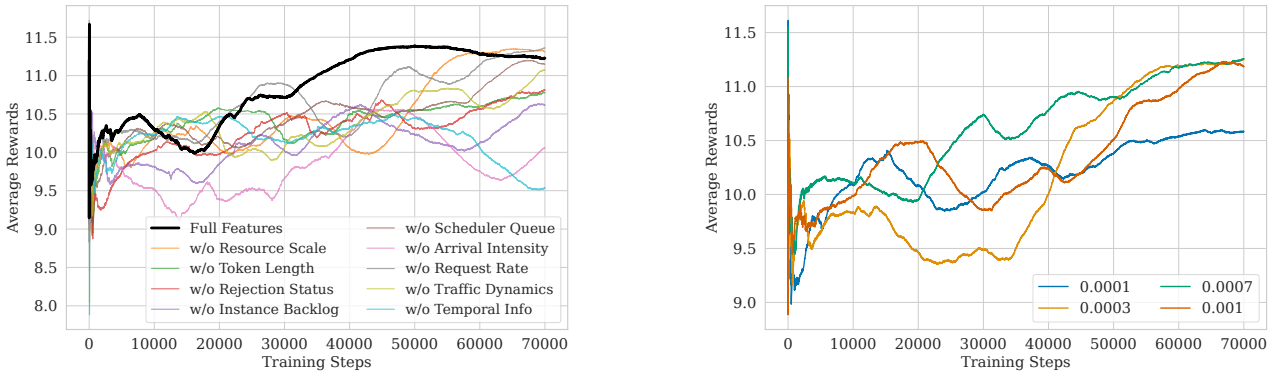
To evaluate system performance under controllable and reproducible conditions, we conducted experiments on a high-fidelity discrete-event simulator based on Splitwise [Patel *et al.*, 2024]. This simulator is specifically tailored for PD disaggregated scenarios in LLM inference, capable of precisely mimicking the dynamic interactions of heterogeneous instance pools.

Our experiments utilize the BurstGPT [Wang *et al.*, 2025] dataset, derived from real-world production workloads of the Microsoft Azure OpenAI GPT service. This dataset is characterized by extreme burstiness and non-stationary arrival patterns. To ensure the authenticity of the evaluation, we replay the requests strictly following the original timestamps to preserve the inter-arrival time distributions. Furthermore, the input prompt and output response token lengths for each request in the simulation are identical to the dataset records, faithfully reconstructing the distribution of long and short texts in real-world business scenarios.The code is publicly available at https://github.com/Onlytonight/SplitScaling.

To validate the effectiveness of our proposed architecture, we compare it against the following two baseline strategies:

Optimal Static Ratio [Qin *et al.*, 2025; Patel *et al.*, 2024]: This strategy statically partitions cluster resources into Prefill and Decode instance pools. To ensure a rigorous comparison, we performed an exhaustive offline grid search across the resource configuration space. As visualized in Fig. 5, we traversed all potential combinations of Prefill and Decode instance counts to identify the Pareto-optimal static configuration. We selected the specific fixed setup that minimizes resource provisioning while satisfying SLO constraints.

HeteroScale [Li *et al.*, 2025]: Representing one of the state-of-the-art solutions for PD disaggregated architectures, HeteroScale uses Decode Tokens Per Second as the core control signal to synchronously scale Prefill and Decode resource pools based on pre-defined control ratios.



(a) Impact of state feature ablation.

(b) Impact of learning rates on convergence.

Figure 4: **Convergence analysis of the DRL agent.** Subfigure (a) demonstrates that comprehensive spatiotemporal state features are essential for achieving high rewards. Subfigure (b) identifies 7×10^{-4} as the optimal learning rate, balancing learning speed with stability.

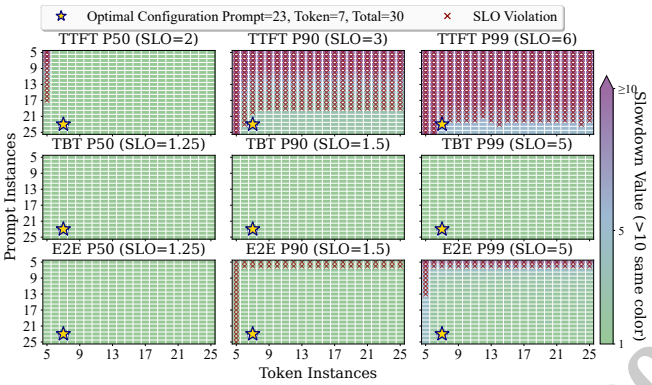


Figure 5: **Heatmap of SLO violations across the configuration space.** We performed an exhaustive search to determine the Optimal Static Ratio baseline, selecting the configuration with minimal resource usage among those satisfying the SLO threshold.

w_c	w_s	TTFT P99	Cost	w_c	w_s	TTFT P99	Cost
0.2	0.8	1.00	118.68	0.5	0.5	2.49	13.88
0.3	0.7	1.37	24.90	0.6	0.4	2.81	10.09
0.4	0.6	1.63	12.43	0.7	0.3	5.99	6.01

Table 1: Performance results under various weight configurations.

5.2 DRL Agent Training and Sensitivity Analysis

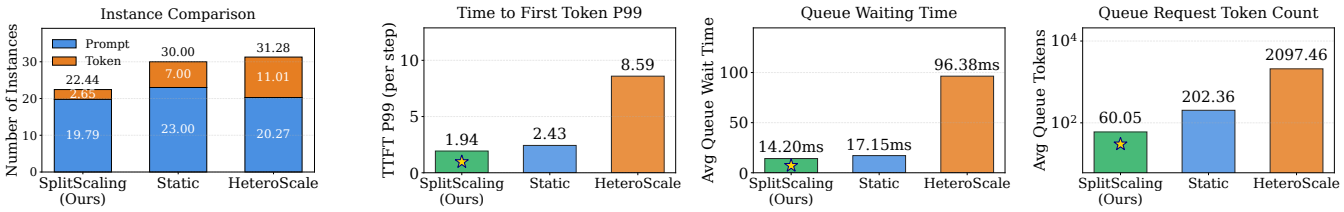
To verify the empirical robustness of the proposed DRL controller, we conducted a sensitivity analysis on state space representation and policy learning rates. Fig. 4 presents the ablation study results for state features. The experiments indicate that the baseline configuration, which includes comprehensive workload features, exhibits the most robust performance, with the final converged reward significantly outperforming other variants. Removing key dimensions such as Instance Backlog, Temporal Info, or Arrival Intensity leads to severe oscillations in the convergence trajectory and degraded reward values. This empirically demonstrates that the DRL agent must rely on fine-grained spatiotemporal features to accurately capture the non-linear mapping of resource demands for Prefill and Decode phases under heterogeneous loads.

Furthermore, Fig. 4 (b) compares the impact of different policy learning rates on convergence behavior. Results show that a learning rate of 7×10^{-4} achieves the optimal balance between learning speed and convergence stability. An excessively high learning rate of 10^{-3}, while initially faster, tends to trap the agent in suboptimal policies, whereas a lower rate fails to adapt quickly enough to transient bursts in production environments. In summary, optimized hyperparameter configuration ensures that the Scaling Controller can precisely respond to complex traffic fluctuations.

Table 1 evaluates the sensitivity of multi-objective reward weights. We observe a critical trade-off: excessive w_c triggers aggressive de-provisioning and SLO violations, whereas disproportionate w_s incurs resource redundancy. The configuration ($w_c = 0.4, w_s = 0.6$) yields the optimal balance, achieving minimal tail latency with high cost-efficiency. This underscores the necessity of calibrated weighting to reconcile service stability with resource utilization.

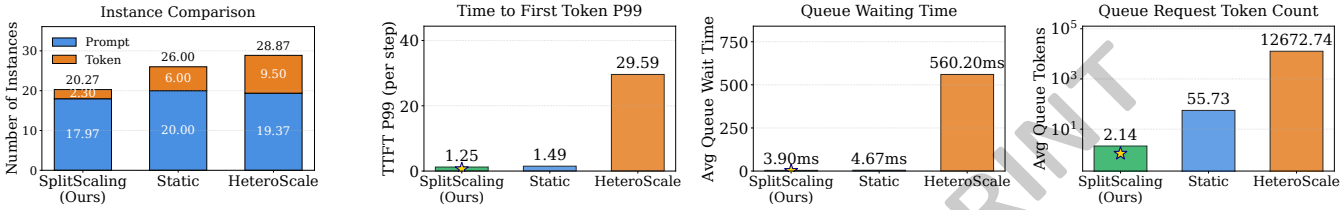
5.3 System Resilience under Extreme Bursty Workloads

We subjected the system to high-intensity bursty traffic to assess the reactive resilience of the auto-scaling strategies. As shown in Fig. 6, SplitScaling significantly outperforms baseline schemes by maintaining superior resource balance. Although HeteroScale attempts to scale, its heuristic-based strategy leads to a severe misalignment between Prefill and Decode resources. This misalignment triggers a catastrophic surge in P99 TTFT slowdown, reaching a factor of 8.59 relative to ideal latency, and causes massive queue backlogs exceeding 2000 request tokens. In contrast, our DRL agent achieves precise and decoupled capacity adjustments. It stabilizes the P99 TTFT slowdown at 1.94 while reducing average queue wait time to 14.20ms. Notably, while guaranteeing high Quality of Service, SplitScaling utilizes only 22.44 average instances. This reduces computational costs by 25.2% and 28.3% compared to the Optimal Static configuration and



(a) Resource usage. (b) TTFT slowdown and queue length.

Figure 6: **Performance comparison under extreme bursty workloads.** (a) SplitScaling significantly reduces resource provisioning compared to Optimal Static and HeteroScale baselines. (b) Our method strictly adheres to SLOs with minimal queue buildup, whereas HeteroScale suffers from catastrophic congestion and latency spikes.



(a) Resource usage. (b) TTFT slowdown and queue length.

Figure 7: **Zero-shot generalization on unseen workloads.** The agent, trained solely on Trace A, demonstrates robust performance transfer when tested on the distinct Trace B. It maintains low TTFT slowdown and efficient resource usage without requiring additional fine-tuning.

HeteroScale respectively. This provides compelling evidence of the agent capability to proactively suppress queue accumulation and maintain system stability under extreme non-stationary demands.

5.4 Generalization Experiments

To evaluate the zero-shot generalization capability of the proposed DRL strategy, we deployed the agent trained on a specific temporal workload into a completely new time-domain scenario with significantly distinct burst patterns. As illustrated in Fig. 7a and Fig. 7b, SplitScaling demonstrates exceptional adaptability, achieving efficient resource savings while strictly adhering to QoS. Specifically, compared to the Optimal Static and HeteroScale baselines, our method reduces the average instance count by 22% and 29.8% respectively. Simultaneously, it achieves the lowest P99 TTFT slowdown of 1.25 and near-zero queue accumulation, averaging only 2.14 request tokens. It is worth noting that HeteroScale suffers a disastrous performance collapse in this unseen scenario, with P99 TTFT slowdown skyrocketing to 29.59. Conversely, our agent maintains stable sub-millisecond queue latency. This indicates that the policy has effectively internalized the underlying non-linear dynamic characteristics of the PD disaggregated architecture enabling robust decision-making. The DRL agent not only transcends the limitations of specific training data distributions but also efficiently handles non-stationary traffic variations.

6 Discussion

In terms of system modeling, our framework incorporates the boot-up delay caused by model loading. This latency results in a reward lag following scale-out operations, which inherently suppresses scaling thrashing. To ensure high availability, we establish hard min/max node limits to constrain the system state within a secure range. Combined with a Connection Draining mechanism, the framework achieves zero service interruption during scale-in. To address potential single-point bottlenecks under high RPS, the Atomic Scheduler supports high-availability deployment through multi-replica redundancy. Regarding system overhead, DRL inference operates at a microsecond level while scaling execution occurs at minute-level intervals; this significant difference in magnitude renders the algorithm's computational overhead negligible.

7 Conclusion

This paper proposes an adaptive auto-scaling framework tailored for disaggregated LLM inference architectures, enabling collaborative management of Prefill and Decode resources under non-linear, dual-bursty loads. By formulating the resource provisioning problem as a MDP and introducing a Just-in-Time Rescheduling mechanism, the system effectively bridges the performance gap caused by scaling lag. Experimental results demonstrate that our solution significantly outperforms existing heuristics in both SLO attainment and cost-efficiency. Future work will focus on stabilization mechanisms to suppress frequent scaling switches (i.e., resource thrashing). Such oscillations incur significant model reloading overheads and redundant energy consumption, potentially undermining system stability and operational robustness. Addressing these fluctuations is critical for building sustainable and reliable large-scale AI infrastructure.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant No. 62372462 and Grant No. U22B2005.

Contribution Statement

Wei Xiao, Xuefeng Huang and Weijia Shi contributed equally to this work and are considered co-first authors. Baokang Zhao is the corresponding author.

References

[Achiam *et al.*, 2023] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[Agrawal *et al.*, 2024] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, 2024.

[Aminabadi *et al.*, 2022] 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.

[Dao *et al.*, 2022] 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.

[Fan *et al.*, 2022] Yuping Fan, Boyang Li, Dustin Favorite, Naunidh Singh, Taylor Childers, Paul Rich, William Allcock, Michael E Papka, and Zhiling Lan. Dras: Deep reinforcement learning for cluster scheduling in high performance computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4903–4917, 2022.

[Fu *et al.*, 2024] 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.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[Hu *et al.*, 2025] CunChen Hu, HeYang Huang, LiangLiang Xu, XuSheng Chen, Chenxi Wang, Jiang Xu, Shuang Chen, Hao Feng, Sa Wang, Yungang Bao, et al. Shuffleinfer: Disaggregate llm inference for mixed downstream workloads. *ACM Transactions on Architecture and Code Optimization*, 2025.

[Kwon *et al.*, 2023] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[Li *et al.*, 2023] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, Boston, MA, July 2023. USENIX Association.

[Li *et al.*, 2024] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*, pages 28935–28948. PMLR, 2024.

[Li *et al.*, 2025] Rongzhi Li, Ruogu Du, Zefang Chu, Sida Zhao, Chunlei Han, Zuocheng Shi, Yiwen Shao, Huanle Han, Long Huang, Zherui Liu, and Shufan Liu. Taming the chaos: Coordinated autoscaling for heterogeneous and disaggregated llm inference, 2025.

[Ma *et al.*, 2024] Xiang Ma, Kexuan Zong, and Amin Rezaeipanah. Auto-scaling and computation offloading in edge/cloud computing: a fuzzy q-learning-based approach. *Wireless Networks*, 30(2):637–648, 2024.

[Mao *et al.*, 2016] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56, 2016.

[Mao *et al.*, 2017] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 197–210, 2017.

[Mao *et al.*, 2019] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, et al. Park: An open platform for learning-augmented computer systems. *Advances in Neural Information Processing Systems*, 32, 2019.

[Miao *et al.*, 2024a] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi,

et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024.

[Miao *et al.*, 2024b] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 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*, pages 1112–1127, 2024.

[Patel *et al.*, 2024] 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.

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

[Rzadca *et al.*, 2020] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. Autopilot: workload autoscaling at google. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[San Juan and Wong, 2023] Justin San Juan and Bernard Wong. Reducing the cost of gpu cold starts in serverless deep learning inference serving. In *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 225–230. IEEE, 2023.

[Sheng *et al.*, 2023] 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.

[Touvron *et al.*, 2023] 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.

[Wang *et al.*, 2025] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize llm serving systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25)*, Toronto, ON, Canada, 2025. ACM.

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

[Xue *et al.*, 2022] Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, et al. A meta reinforcement learning approach for predictive autoscaling in the cloud. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4290–4299, 2022.

[Yu *et al.*, 2022] 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, Carlsbad, CA, July 2022. USENIX Association.

[Zhang *et al.*, 2025] Zeyu Zhang, Haiying Shen, Shay Vargaftik, Ran Ben Basat, Michael Mitzenmacher, and Minlan Yu. Hack: Homomorphic acceleration via compression of the key-value cache for disaggregated llm inference. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 1245–1247, 2025.

[Zheng *et al.*, 2024] 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. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

[Zhong *et al.*, 2024] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation*, OSDI'24, USA, 2024. USENIX Association.