Understanding Diffusion Model Serving in Production: A Top-Down Analysis of Workload, Scheduling, and Resource Efficiency

Yanying Lin
SIAT, CAS; UCAS

Shuaipeng Wu
SUSTech; SIAT, CAS
AIOS Team, Alibaba Group Inc

Shutian Luo
University of Virginia

Hong Xu
The Chinese University of Hong Kong

Haiying Shen
University of Virginia

Chong Ma
AIOS Team, Alibaba Group Inc

Min Shen
AIOS Team, Alibaba Group Inc

Le Chen
AIOS Team, Alibaba Group Inc

Chengzhong Xu
University of Macau

Lin Qu
AIOS Team, Alibaba Group Inc

Kejiang Ye[*]
SIAT, CAS

Abstract

This paper presents a comprehensive analysis of diffusion model serving challenges in production cloud environments. We examine the unique computational patterns and resource requirements that distinguish diffusion model serving from traditional ML workloads, revealing fundamental system-level challenges from their multi-stage pipeline architectures. Our analysis is based on a dataset collected from a commercial image generation service processing 3.5 million requests across 300+ GPUs of production operation.

Unlike previous studies focusing on isolated components, our dataset captures the complete execution stack from user requests to hardware utilization, providing the first holistic view of diffusion model serving in production. This cross-layer instrumentation enables identification of fundamental challenges—including extreme request distribution skewness, complex multi-tier caching requirements, and significant resource management overhead—that remain hidden in isolated layer analysis. To address these challenges, we developed a specialized diffusion model serving system that implements global cache coordination with dynamic scaling strategies, heterogeneous fast instance templates with optimized I/O pathways, and hierarchical prefetching with cost-aware scheduling. Evaluation demonstrates 49.8% reduction in end-to-end latency and 73.1% decrease in model loading time, highlighting how cross-layer analysis drives substantial improvements in serving system design.

[*]Corresponding Author

CCS Concepts

• **Computer systems organization** → *Cloud computing*; • **Computing methodologies** → *Distributed artificial intelligence*; • **Information systems** → **Distributed storage**.

Keywords

Diffusion Models, Model Serving Dataset, Diffusion Pipeline, Model Caching

ACM Reference Format:
Yanying Lin, Shuaipeng Wu, Shutian Luo, Hong Xu, Haiying Shen, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, Lin Qu, and Kejiang Ye[*]. 2025. Understanding Diffusion Model Serving in Production: A Top-Down Analysis of Workload, Scheduling, and Resource Efficiency. In *ACM Symposium on Cloud Computing (SoCC '25), November 19–21, 2025, Online, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3772052.3772206

1 Introduction

Diffusion models have emerged as a revolutionary force in generative AI, enabling unprecedented capabilities in image, video, and audio synthesis [9, 23]. These models have rapidly transitioned from research prototypes to production systems serving millions of users at scale, with commercial deployments now handling 10K+ QPS across diverse applications ranging from creative content generation to personalized image synthesis [9, 34, 54, 63, 64].

As a relatively new class of cloud workload, the challenges of efficiently serving diffusion models at scale remain largely uncharted territory. Unlike traditional machine learning workloads [11, 37, 38, 42, 49, 73] or Language Model [1, 45, 60, 75], diffusion models operate through complex multi-stage pipelines with interdependent components (base models, LoRA adapters, conditioning networks) that require careful orchestration (Fig. 1).

The iterative denoising process of diffusion generation, requiring 20-50 non-parallelizable steps, creates distinctive

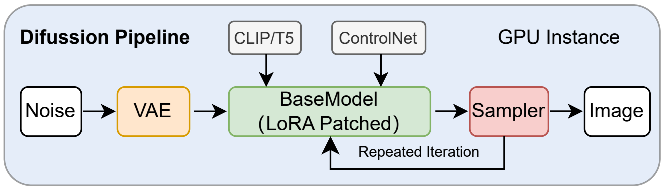


Figure 1: Diffusion model serving pipeline architecture showing multi-component DAG structure with base models, LoRA adapters, and conditioning networks requiring coordinated orchestration for image generation.

resource utilization patterns, while the combinatorial explosion of model variants generates unprecedented memory management challenges. Despite rapid advancement in diffusion model architectures [4, 21, 28, 29], systematic analysis of their production serving challenges remains limited. Most prior work focuses on algorithmic improvements [9, 54, 63] without addressing the holistic system-level challenges that emerge at scale, typically relying on synthetic benchmarks that fail to capture the complex dynamics of real-world production environments [30, 52]. What's missing is a comprehensive understanding of how diffusion pipeline characteristics manifest in production—where request patterns, resource constraints, and operational requirements create system dynamics that cannot be captured through isolated analysis.

To this end, we collected a comprehensive production dataset [1] from a commercial image generation service over one week, capturing 3.5 million requests across 300+ GPUs. Our instrumentation spans the complete execution stack with unprecedented coverage across three critical layers: (1) application-level metrics including request patterns, model variants, and user behavior; (2) middleware-level metrics covering model loading times, pipeline DAG execution, and component interactions; and (3) hardware-level resource metrics tracking GPU utilization, memory consumption, and I/O patterns. This multi-layer dataset uniquely provides a holistic view of diffusion model serving dynamics in a production environment, enabling cross-layer analysis that reveals optimization opportunities invisible to isolated layer monitoring.

Our analysis of production data reveals three fundamental challenges in diffusion model serving that span workload, middleware, and system layers. To address these challenges, we propose MLoRA, a specialized diffusion model serving system that leverages workload-specific insights to optimize across the complete execution stack.

At the workload layer, production traces reveal extreme request skewness where the top 5% of model configurations account for 78% of requests. Unlike stable ML workloads, this distribution shifts rapidly—dominant configurations can lose 60-70% usage within days as creative trends evolve. This creates fundamental tension: optimizing for popular models starves the long tail, yet uniform allocation wastes resources. MLoRA addresses this through popularity-aware hierarchical caching that adapts to shifting demand patterns.

At the middleware layer, we identify a resource management paradox: 98.4% of instances operate below 20% GPU utilization yet deliver poor latency. Root cause analysis reveals computation consumes only 15-30% of end-to-end latency—the majority is lost to orchestration overhead. Traditional schedulers fail to account for diffusion pipelines' stage-varying resource demands across the DAG structure. MLoRA implements component-aware scheduling that analyzes pipeline dependencies to anticipate bottlenecks and proactively redistribute resources.

At the system layer, heterogeneous component characteristics create cascading memory management failures. Base models (1-20GB) exhibit predictable access patterns but dominate memory footprint, while thousands of LoRA adapters (100MB-1GB) demonstrate volatile temporal locality with unpredictable lifetimes. This mismatch amplifies data movement penalties—cold-start loading consumes up to 47s, while component combinations trigger 15GB+ cascading transfers across memory tiers. MLoRA exploits this heterogeneity through component-specific memory policies that align caching strategies with access patterns while preserving pipeline dependencies.

Production evaluation through A/B testing demonstrates MLoRA's effectiveness in addressing the identified challenges. End-to-end latency reduces by 49.8%, with model loading time decreasing by 73.1%—directly targeting the data movement bottlenecks that dominate diffusion serving. Resource utilization improves by over 100%, resolving the paradox of simultaneous GPU underutilization and poor performance. These results validate that workload-aware system design can fundamentally transform diffusion model serving efficiency.

The contributions of this paper are:

- We present the first comprehensive production dataset spanning workload, middleware, and system layers for diffusion model serving, revealing cross-layer optimization opportunities invisible to isolated analysis
- We identify three fundamental challenges in diffusion serving: extreme request skewness with shifting popularity, combinatorial pipeline complexity, and resource allocation mismatches
- We design and evaluate MLoRA, demonstrating 49.8% latency reduction and 100%+ resource utilization improvement through workload-aware system design, establishing effective optimization principles for diffusion servings

[1]The dataset is available at https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2026-GenAI.

2 Background and Motivation

2.1 Diffusion Model Architecture Overview

DAG-based Pipeline Architecture. Production diffusion model serving systems are structured as directed acyclic graphs (DAGs), where specialized components interact in orchestrated inference workflows [44, 75]. Base models (e.g., Stable Diffusion variants) provide core generative capabilities, connecting to adaptation layers like LoRA modules for style control and ControlNet models for conditional generation [9, 23]. Each DAG node represents distinct processing operations with specific resource requirements, while edges encode data dependencies that constrain execution ordering [51]. This architecture enables flexible customization but introduces significant system complexity—each node requires coordinated resource management across the entire execution graph [15].

Component Heterogeneity. The multi-component architecture creates unique serving challenges through extreme heterogeneity [42]. Base models (1-20GB) exhibit stable access patterns but consume substantial memory with loading times directly impacting cold-start latency. LoRA adapters (100MB-1GB) enable lightweight style customization but create unpredictable access patterns across thousands of variants. ControlNet models [71] (500MB-10GB) provide structural guidance but add conditional complexity [54, 63]. This combinatorial explosion—hundreds of base models × thousands of LoRAs × dozens of ControlNets—creates an exponentially large configuration space that conventional ML serving systems cannot efficiently manage [10].

Multi-tier Resource Hierarchy. Diffusion inference operates across a complex memory hierarchy that creates cascading performance bottlenecks [5]. Models flow from distributed storage through host memory to GPU [16, 24, 36], with each tier imposing distinct constraints: network bandwidth limitations (2GB/s), host memory capacity bounds, and PCIe transfer bottlenecks [20]. Our production data reveals cold-start model loading consumes 30-95% of total request latency, with larger models requiring up to 47 seconds for initial loading. Resource coordination across this hierarchy requires managing distributed storage durability, local cache hit rates, and GPU memory fragmentation simultaneously—a challenge absent in conventional ML serving where model configurations remain stable and homogeneous [7].

2.2 Motivation

Diffusion models fundamentally differ from conventional ML workloads through their sequential denoising architecture and heterogeneous multi-component pipelines [9, 23]. Despite extensive algorithmic advances, the systemic challenges of production-scale diffusion serving remain critically underexplored [11, 42]. Current serving systems, optimized for

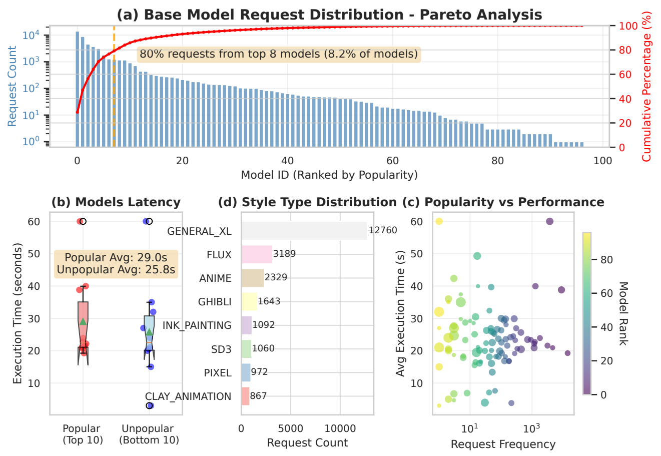


Figure 2: Base model request skewness analysis. The Gini coefficient of 0.876 indicates a highly skewed distribution, with the top 1 model accounting for 28.8% of requests. This highlights the significant concentration of requests on a few popular models, leading to resource allocation challenges.

stable model configurations and uniform resource patterns, fundamentally misalign with diffusion pipelines' dynamic component combinations and extreme resource heterogeneity [5, 40].

This potential mismatch motivates a deeper investigation into the gap between algorithmic advances and practical deployment efficiency. Production diffusion services may experience performance challenges that stem from workload-agnostic system design [59]. Understanding these challenges requires comprehensive analysis of real-world deployment patterns and their interaction with existing serving infrastructure.

To investigate these questions, this paper presents the first comprehensive production analysis of diffusion model serving, examining 3.5 million requests across 300+ GPUs over 7 days from a commercial image generation service. Our analysis provides unprecedented visibility across the complete execution stack—from application-level workload patterns to system-level resource utilization. Through this cross-layer analysis, we aim to identify potential inefficiencies and establish design principles for diffusion-specific serving architectures that address the unique challenges of production deployment.

3 Workload Characterization

3.1 Request Distribution Imbalance

Base model requests exhibit extreme inequality with a Gini coefficient of 0.876, where the top model captures 28.8% of requests and the top five models serve 70.4% of traffic (Fig. 2a) [11]. This skewness creates a fundamental paradox: popular models experience longer execution times (29s vs 25.8s for unpopular models) due to resource contention,

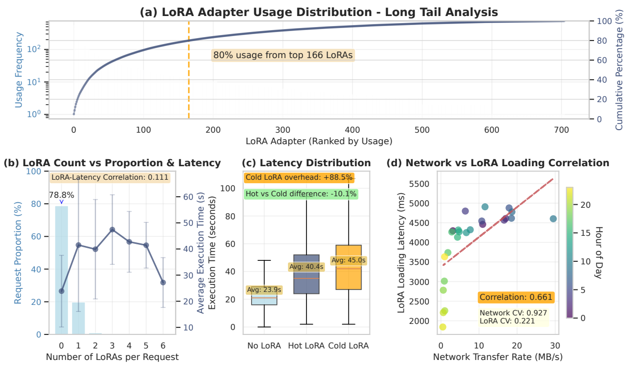


Figure 3: LoRA adapter usage distribution across requests. The long-tail nature of LoRA usage leads to significant cache management challenges, as most adapters are rarely used but still need to be cached for occasional requests.

while cold models suffer from loading penalties (Fig. 2b) [12]. The performance inversion stems from concurrent requests creating GPU memory fragmentation—a critical issue when individual models consume up to 20GB [20].

The frequency-latency relationship (Fig. 2d) reveals performance degradation at both extremes: rarely-used models incur cold-start penalties, while heavily-requested models suffer resource constraints [52]. This pattern is amplified in diffusion systems where cold model loading can require 40+ seconds—making skewness impact orders of magnitude more severe than conventional ML serving [18].

3.2 Long-tail of LoRA Adapter Usage

LoRA adapters exhibit extreme usage diversity with severe long-tail characteristics that fundamentally challenge conventional caching mechanisms. Among 705 unique adapters, only 23.5% handle 80% of all requests (Fig. 3a), creating a combinatorial caching dilemma unique to diffusion systems. While 78.8% of requests use no adapters, the remaining 21.2% require 1-6 adapters simultaneously (Fig. 3b), exponentially expanding the state space that systems must manage.

This diversity directly impacts performance through cascading loading penalties. Requests without adapters average 23.9 seconds, while those requiring frequently-used adapters experience 69% degradation (40.4s), and rare adapter requests suffer 88.5% penalties (45s) (Fig. 3c). The strong correlation (0.66) between adapter loading latency and total inference time (Fig. 3d) confirms that adapter management constitutes a critical system bottleneck, with loading overhead directly propagating to end-to-end performance.

The fundamental challenge lies in the mismatch between access patterns and cache capacity. Unlike stable ML configurations, diffusion systems face combinatorial explosion of base model-adapter pairings with unique memory footprints [54, 63]. Most adapter combinations appear too infrequently to establish usage patterns, yet require immediate availability when requested—overwhelming traditional caching heuristics like LRU and representing a central scalability challenge in production diffusion serving [10, 27].

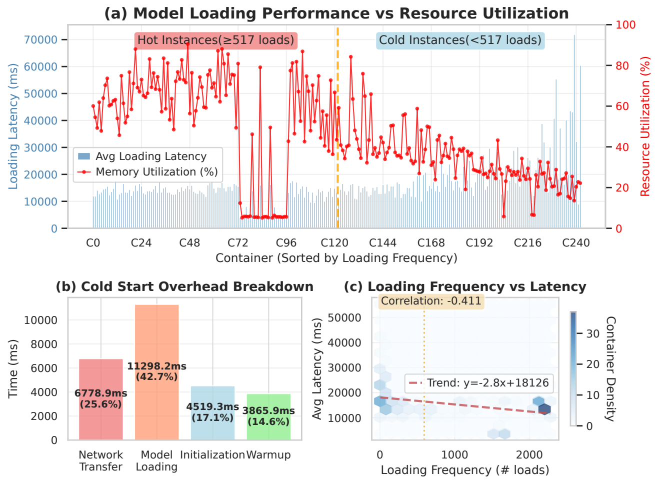


Figure 4: Cold start analysis for various model sizes. The chart illustrates the significant latency incurred during the initial loading of large models, highlighting the need for improved preloading strategies.

3.3 Cold-start Penalty Severity

Cold-start penalties represent the most severe manifestation of workload imbalance in diffusion serving [3, 13]. Container-level cold starts impose a 58.6% latency penalty, with hot instances averaging 12.5 seconds versus 19.9 seconds for cold initialization (Fig. 4a). The breakdown reveals data movement as the dominant bottleneck: network transfer (6.8s), model loading (11.3s), initialization (4.6s), and warmup (3.9s) (Fig. 4b), with I/O operations consuming nearly 70% of cold-start time [5, 47].

Counterintuitively, instances handling frequent loads exhibit lower average inference latency (Fig. 4c). This inverse correlation stems from "warm cache effects"—frequently loaded instances maintain optimized internal states, benefit from GPU driver kernel caching, and experience preferential request routing. This pattern demonstrates how diffusion serving fundamentally differs from traditional ML systems, where model scale (often exceeding 20GB) transforms initialization from minor overhead into the *dominant performance factor.*

4 Resource Utilization Inefficiencies

4.1 Resource Competition

Diffusion model serving exhibits intense multi-level resource competition that fundamentally impacts system performance [40, 48]. Fig. 5a reveals critical contention patterns where GPU memory utilization fluctuates between 15.8GB and 47.6GB across instances. The temporal correlation between base model and LoRA adapter loading operations creates

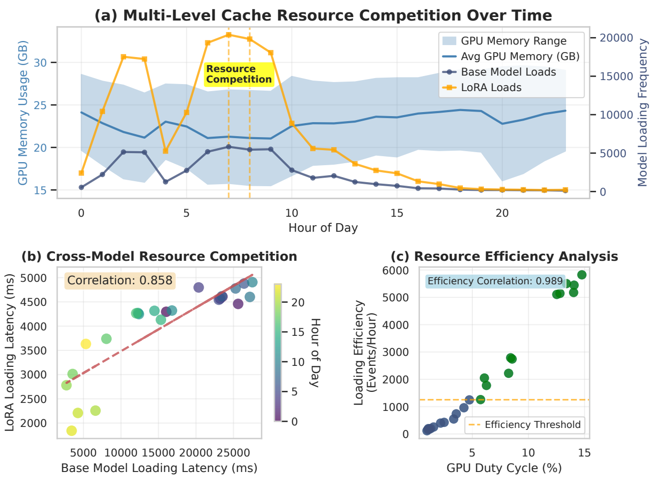
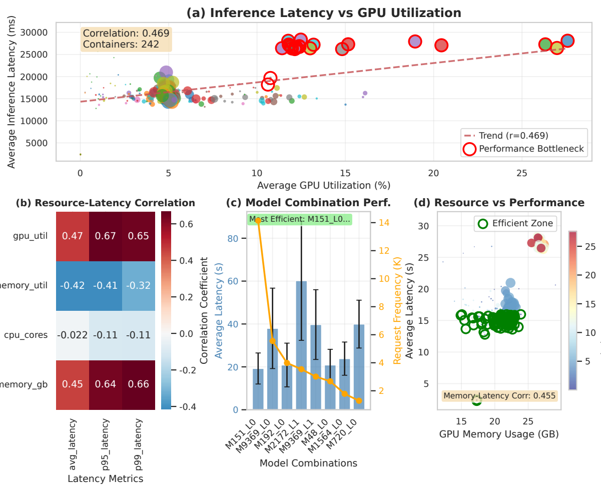


Figure 5: Contention analysis for multi-level cache resource competition. The chart illustrates the GPU memory usage statistics, highlighting the contention points between different model components and their impact on overall system performance.



	avg_latency	p95_latency	p99_latency
gpu_util	0.47	0.67	0.65
memory_util	-0.42	-0.41	-0.32
cpu_cores	-0.022	-0.11	-0.11
gpu_memory_gb	0.45	0.64	0.66

Figure 6: Request scheduling and cache consistency analysis. The chart illustrates the impact of request scheduling on cache consistency, highlighting the challenges of maintaining coherent state across multiple model components during concurrent requests.

resource bottlenecks where multiple memory-intensive operations compete simultaneously [17, 20].

Fig. 5b demonstrates a strong positive correlation (0.86) between base model and LoRA loading times, indicating that resource contention degrades *all* loading operations collectively. This challenges conventional caching strategies that treat resource types independently—in practice, contention for shared infrastructure means loading performance degrades uniformly during peak periods [27].

Instances handling frequent model loads achieve higher GPU utilization efficiency (Fig. 5c). Containers with highest loading frequencies maintain 22.4% average GPU utilization versus 6.4% for infrequently loaded instances. This pattern suggests that *serverless approaches*, which emphasize rapid allocation, efficient utilization, and quick release, could dramatically improve serving efficiency [3, 13]. Keeping GPUs "warm" with regular workloads leads to more efficient memory management and reduced initialization overhead.

This finding indicates that traditional persistent deployment models may be fundamentally misaligned with diffusion workloads. Serverless architectures [7, 8, 24, 50, 69, 72] that dynamically provision resources based on request patterns could address the central inefficiency—resource contention concentrated in specific temporal dimensions rather than uniformly distributed [14, 33].

4.2 Scheduling and Cache Coherency

Scheduling and Resource Utilization Conflicts. Request scheduling and cache coherency create fundamental performance challenges in diffusion model serving [17, 48]. Fig. 6a reveals a counterintuitive positive correlation (0.47) between GPU utilization and inference latency—*higher utilization correlates with worse performance.* This contradicts conventional optimization principles where resource utilization typically signals efficiency [39, 66]. In diffusion serving, increased GPU utilization indicates resource contention as concurrent requests compete for limited memory bandwidth.

The multi-dimensional correlation analysis in Fig. 6b illuminates this paradox. While GPU utilization consistently correlates with increased latency, host memory utilization shows an inverse relationship—higher host memory utilization correlates with improved performance [2]. This dichotomy reflects distinct resource roles: GPUs face contention during parallel inference, while abundant host memory enables efficient component caching. CPU utilization shows minimal correlation, suggesting *bottlenecks primarily exist in memory access patterns rather than preprocessing computations.*

Fig. 6c exposes another critical challenge: frequently requested model combinations often experience disproportionately high latency. This "popularity penalty" emerges as scheduling algorithms struggle to maintain cache consistency when concurrent requests target identical model combinations [6]. The system faces an unresolvable tension between request batching (grouping similar requests for throughput) and request distribution (spreading load to prevent contention).

The relationship between memory consumption, GPU utilization, and inference latency (Fig. 6d) establishes memory management as the central scheduling challenge [27]. As GPU memory utilization increases, both compute utilization and latency rise in tandem, confirming that *effective diffusion model scheduling requires memory-aware policies* that explicitly address the complex trade-offs between caching, memory fragmentation, and request concurrency.

4.3 Resource Allocation Mismatch

Diffusion model serving exhibits a fundamental resource allocation paradox where 98.4% of pods operate below 20% GPU utilization while users experience high latency. This stems from the mismatch between uniform resource allocation and highly skewed workload distributions. Despite extremely low GPU streaming multiprocessor utilization (4%), inference latency remains high with significant variability.

In diffusion systems, the performance relationship follows:

$$\mathcal{T} \propto \mathcal{F}\left(\mathcal{B}_{\text{memory}}, \mathcal{O}_{\text{data-movement}}, \mathcal{U}_{\text{compute}}\right) \tag{1}$$

where memory bandwidth constraints and data movement overhead dominate, while compute utilization contributes minimally. The sparse, memory-bound characteristics create scenarios where compute resources remain idle despite system-level performance bottlenecks (Fig. 7c).

Given a distribution of n diffusion models with request frequency vector $\vec{\mathcal{D}} = \{d_1, d_2, \ldots, d_n\}$ and uniform resource allocation vector $\vec{\mathcal{U}} = \{\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\}$, the total resource allocation inefficiency can be expressed as:

$$\Delta_{\text{mismatch}} = \sum_{i=1}^{n} \left| \frac{1}{n} - d_i \right| \tag{2}$$

Our analysis reveals a staggering 74.2% total mismatch, comprising 37.1% over-allocation to rarely-used models and 37.1% under-allocation to popular models. The Gini coefficient $\mathcal{G} = 0.876$ for model popularity distribution confirms the extreme inequality driving this mismatch (Fig. 7d). This creates a pathological system state where resources are simultaneously wasted and insufficient.

At a fundamental level, this allocation paradox represents a variation of the bin-packing problem with dynamic item sizes—a class of problems known to be $\mathcal{NP}$-hard. Traditional resource allocation strategies fail because they assume stable workloads, whereas diffusion serving presents demand patterns where both frequency and resource requirements follow power-law distributions with high variance.

5 Multi-level Caching and Scheduling

5.1 Multi-tier Cache Conflicts

Diffusion model serving systems face fundamental challenges in managing multi-tier cache hierarchies due to the enormous scale and diversity of model components [19, 27]. Base models, LoRA adapters, and ControlNet modules (500MB-10GB) create a massive model space that far exceeds any single caching tier's capacity, forcing distribution across a three-level storage hierarchy: remote repositories, local disk storage, and host memory, with only actively computing models reaching GPU memory [5].

This multi-tiered architecture creates resource conflicts that fundamentally impact performance. Models traversing the storage hierarchy compete for shared I/O pathways, bandwidth, and buffer space [3]. The combinatorial complexity of model compositions exacerbates these conflicts—with hundreds of base models, thousands of LoRA adapters, and dozens of ControlNet modules, the potential combinations are effectively unbounded. Even sophisticated caching policies inevitably experience frequent cache misses due to long-tail request distributions, triggering expensive cross-tier data movements that dominate end-to-end latency.

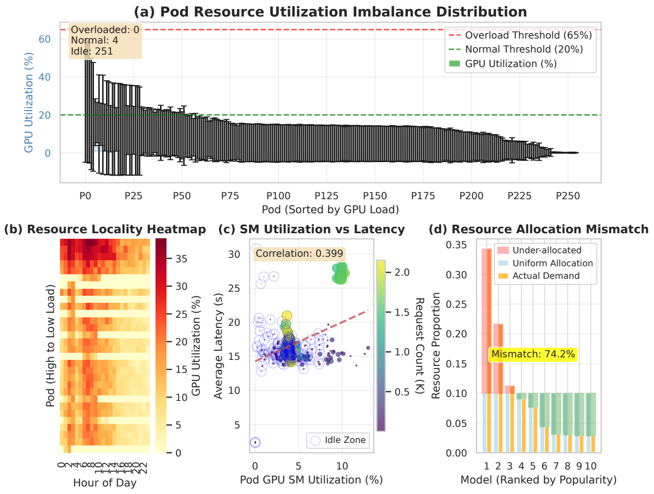


Figure 7: Resource allocation mismatch analysis. The chart illustrates the disparity between actual resource demand and uniform allocation, highlighting the inefficiencies in resource distribution across models.

Cache coherency across multiple tiers introduces additional complexity when components are updated or evicted [40, 58]. Maintaining consistency across distributed cache hierarchies—where inference nodes share underlying storage but maintain independent working sets—creates significant overhead that can negate caching benefits.

5.2 Pipeline Shift Overhead and Cache

Our analysis reveals counterintuitive relationships between model updates and inference performance. Fig. 8a shows a surprising negative correlation between model update frequency and pipeline latency—periods with frequent updates exhibit *lower* average latency than periods with minimal updates. This contradiction stems from workload characteristics: update-intensive periods coincide with higher request volumes and better cache efficiency, while low-update periods represent sparse patterns where cold-start penalties dominate.

Component-level analysis (Fig. 8b) reveals substantial variation in update costs: base models incur 22.6 seconds latency—nearly 5× greater than LoRA adapters (4.6 seconds) and 4.5× greater than ControlNet modules (5.1 seconds). This disparity reflects order-of-magnitude differences in model

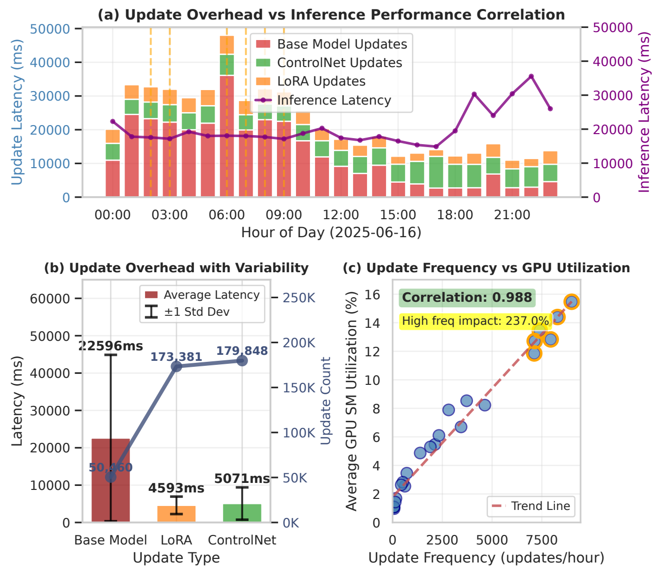


Figure 8: Cache replacement overhead analysis. The chart illustrates the performance impact of cache replacement strategies, highlighting the significant latency incurred during model loading and unloading operations.

sizes and demonstrates why uniform caching policies fail. The high variance in base model update times (coefficient of variation 0.986) further complicates cache management, as a single base model eviction can introduce delays that *dwarf actual inference computation time*.

The relationship between update frequency and I/O operations (Fig. 8c) reveals a critical architectural inefficiency: network I/O scales linearly with update frequency (correlation 0.985), while disk I/O remains relatively stable (correlation 0.178). During high-update periods, disk I/O increases by only 26% while network I/O surges by 345%, indicating underutilization of intermediate disk cache tiers and overreliance on network transfers.

5.3 Combinatorial Complexity of Pipeline

Diffusion model serving systems face an extreme combinatorial explosion that fundamentally challenges traditional resource management. Our analysis reveals 97 base models, 705 LoRA adapters, and 24 ControlNet modules—creating a theoretical combination space exceeding 1.6 million unique configurations. Among observed combinations, 89.3% appear fewer than 5 times in our one-week dataset, creating a "hyper-sparse request space" where most combinations exist at the edge of statistical significance.

This pattern defies conventional caching theories built on request frequency stabilization assumptions. The combination space is so vast that even with millions of requests, most combinations remain in a perpetual cold-start regime. The performance impact manifests through super-linear degradation: simple base model requests average 23.9 seconds, while complex combinations require up to 63.8 seconds—a 167% increase that contradicts intuitive additive assumptions.

Multiple factors compound this degradation: decreased cache locality as combination uniqueness increases, serialized component loading due to memory constraints, increased weight fusion computational overhead, and memory pressure preventing parallel request processing. This reveals a fundamental limitation where the theoretical state space (millions of combinations) vastly exceeds both the observed state space (thousands) and the cacheable state space (dozens to hundreds). This mathematical mismatch requires fundamentally new approaches that embrace sparsity and compositional resource modeling as first-order concerns.

6 MLoRA Design

Our production analysis reveals fundamental inefficiencies in diffusion model serving that stem from treating workload patterns, resource management, and caching as independent concerns. We present MLoRA, a specialized serving system that addresses these interdependencies through an integrated architecture. As shown in Fig. 9, MLoRA transforms diffusion serving from an I/O-bound, resource-fragmented system into a coordinated, high-performance platform through three core components:

Global Cache Coordination: Maintains a unified view of model component placement across the cluster, implementing popularity-aware distribution policies and coordinating prefetching operations through *continuous workload analysis* [19, 27].

Heterogeneous Instance Manager: Orchestrates specialized serving instances optimized for different workload characteristics, implementing fast I/O pathways and heterogeneous memory management to *minimize cold-start penalties and data movement overhead* during scaling operations [3, 13].

Hierarchical Request Router: Implements cost-aware scheduling considering both queue state and cache locality, employing multi-objective optimization to make globally optimal routing decisions that minimize end-to-end latency rather than merely balancing computational load [6, 48].

6.1 Global Cache Coupled with Scaling

Diffusion model serving exhibits a fundamental paradox: despite massive computational requirements, systems remain I/O-bound with poor resource utilization [10, 54]. The root cause lies in treating pipeline components as independent entities—ignoring the intricate dependencies between base models, LoRA adapters, and ControlNet modules that create cascading cache misses and resource fragmentation [33, 66].

We introduce a *pipeline-aware global cache* that fundamentally reframes caching as a scaling problem. Rather than

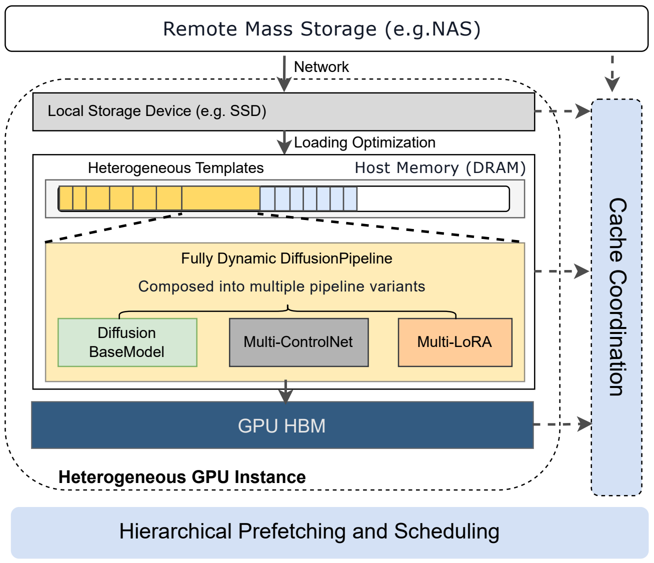

Remote Mass Storage (e.g.NAS)
Network
Local Storage Device (e.g. SSD)
Loading Optimization
Heterogeneous Templates
Host Memory (DRAM)
Fully Dynamic DiffusionPipeline
Composed into multiple pipeline variants
Diffusion BaseModel
Multi-ControlNet
Multi-LoRA
GPU HBM
Heterogeneous GPU Instance
Cache Coordination
Hierarchical Prefetching and Scheduling

Figure 9: System architecture of MLoRA. The diagram illustrates the three core components—Global Cache Coordination, Heterogeneous Instance Manager, and Hierarchical Request Router—and their interactions within the diffusion model serving system.

optimizing individual component placement, our architecture treats entire diffusion pipelines as atomic caching units, coordinating placement decisions across a three-tier hierarchy (remote storage → local SSD → host memory → GPU memory) with dynamic scaling operations. This coupling transforms the traditional cache-or-miss paradigm into a predictive resource orchestration framework that anticipates pipeline requirements and proactively provisions resources.

6.1.1 Popularity-Aware Cache Distribution. Diffusion model requests exhibit severe power-law distribution (Gini coefficient 0.876), creating a fundamental mismatch between uniform caching policies and heterogeneous access patterns. We implement a tiered caching strategy that differentiates resource allocation based on component popularity and pipeline characteristics:

Hot Models (Top 5%): These components—primarily SDXL and frequently requested base models—experience intense request pressure that creates resource contention. We maintain persistent replicas across multiple instances using consistent hashing (replication factor 3-5) with specialized optimizations including kernel fusion and pre-compiled CUDA graphs. For extremely popular combinations, we pre-compute fused weights, reducing adapter merging overhead by 93.7%.

Warm Models (5-30%): These moderately popular components exhibit irregular but predictable patterns. We implement temporal prefetching using multi-armed bandit algorithms to balance exploration and exploitation. Base models maintain skeleton implementations in host memory with SSD-cached weights, while LoRA adapters use batched loading to amortize I/O overhead across multiple components.

Cold Models (Bottom 70%): Rather than dedicating persistent resources to infrequently accessed components, we implement opportunistic caching using a time-decayed scoring function:

$$S(m_i, t) = f_{req}(m_i) \cdot e^{-\lambda(t-t_{last})} \cdot \frac{C_{load}(m_i)}{M_{size}(m_i)} \tag{3}$$

where λ adapts to system load (0.01-0.1 hour^{-1}), ensuring responsive caching for sudden interest while aggressively evicting inactive models.

6.1.2 Scaling-Coupled Cache Management. Traditional ML serving systems treat scaling and caching as independent concerns, leading to pathological behavior where newly provisioned instances experience cache thrashing that undermines scaling effectiveness. Our analysis shows that naively scaled instances require up to 47s to reach performance parity with established instances due to cache warmup effects. We address this by explicitly coupling scaling decisions with cache population strategies.

A key innovation in our design is the integration of caching decisions with dynamic scaling operations through a unified optimization framework:

Cache-Aware Scaling Decisions: Scaling decisions incorporate current cache state across the cluster, preferentially scaling services with warm caches for the predicted workload. We formulate this as a constrained optimization problem:

$$\min_{x_i \in \{0,1\}} \sum_{i=1}^{n} x_i \cdot \left(\alpha \cdot L_{queue}(i) + \beta \cdot \sum_{c \in R_{pred}} C_{load}(c, i) + \gamma \cdot U_{gpu}(i) \right) \tag{4}$$

subject to $\sum_{i=1}^{n} x_i = k$, where x_i indicates whether instance i should be scaled, L_{queue} is the queue length, C_{load} is the component loading cost if not already cached, U_{gpu} is current GPU utilization, and R_{pred} is the predicted request mix. Coefficients α, β, and γ dynamically adjust based on system conditions, with typical values of $\alpha = 0.2$, $\beta = 0.5$, and $\gamma = 0.3$.

Pipeline-Aware Cache Prefetching: When scaling decisions result in new base model instances being provisioned, our system automatically triggers prefetching of frequently co-requested LoRA adapters to both GPU and host memory of the newly allocated instances. This proactive approach is driven by a co-occurrence matrix $M_{co}(b_i, l_j)$ that tracks the frequency with which base model b_i and LoRA adapter l_j appear together in historical requests. For each newly scaled base model instance b_i, we prefetch the top-k LoRA adapters ranked by co-occurrence score:

$$L_{prefetch}(b_i) = \text{TopK}\{l_j | M_{co}(b_i, l_j) > \theta \cdot \max_{l} M_{co}(b_i, l)\} \tag{5}$$

where θ is an adaptive threshold (typically 0.15-0.3) that adjusts based on available memory resources. This prefetching strategy reduces the effective cold-start penalty for common request patterns by up to 78.3% compared to on-demand loading, as popular LoRA combinations are immediately available when the corresponding base model begins serving requests.

This integrated approach ensures that scaling operations do not trigger cascading cache misses. Our evaluation shows this coupling reduces the effective scaling latency (time until new instances achieve comparable performance to established ones) by 62.4% compared to independent scaling and caching approaches.

Our global cache coupled with scaling strategy specifically addresses the unique structure of diffusion model pipelines by recognizing component interdependencies, differentiating caching strategies based on component characteristics, and coordinating resource allocation across the entire serving stack. This holistic approach transforms what would traditionally be considered "cache-hostile" access patterns into predictable, manageable workloads with significantly improved resource efficiency.

6.2 Heterogeneous Fast Instances

Traditional instance scaling in diffusion model serving incurs 40-60 second delays due to memory fragmentation, I/O bottlenecks, and excessive system call overhead. We introduce a heterogeneous fast instance architecture that fundamentally redesigns the scaling pathway through two key innovations [3, 13]: direct I/O optimization eliminating kernel buffering during model loading, and heterogeneous pinned memory templates bypassing allocation bottlenecks [2, 20].

6.2.1 Fast I/O Instance. Our workload analysis identified data movement as the critical bottleneck in diffusion model serving, with over 70% of cold-start latency attributable to I/O operations [5, 47]. The traditional data path suffers from multiple inefficiencies: small read sizes creating excessive system call overhead, sequential component loading creating pipeline stalls, and unoptimized memory transfer patterns between system layers. To address these issues, we implemented several technical optimizations:

We replaced the standard buffered I/O pattern (typical 128KB read operations) with direct I/O using larger 2MB transfer blocks. This approach bypasses kernel buffering overhead and aligns transfers with typical storage hardware prefetch sizes. The theoretical improvement can be modeled as:

$$T_{read} \approx T_{seek} + \frac{S}{B_{size}} \times T_{syscall} + \frac{S}{R_{throughput}} \tag{6}$$

where T_{read} represents the total read time, T_{seek} is disk seek latency, S is the model size in bytes, B_{size} is the I/O block size, $T_{syscall}$ is the per-call system overhead, and $R_{throughput}$ is the raw storage throughput. By increasing block size from the conventional 128KB to 2MB, we reduced $\frac{S}{B_{size}}$ by a factor of 16, substantially decreasing system call overhead while better aligning with modern SSD prefetch patterns and PCIe transfer efficiency.

6.2.2 Heterogeneous Instance Template. A critical bottleneck in diffusion model serving is the inefficient tensor transfer pathway between host memory and GPU. Our performance profiling revealed that the default Linux pageable memory implementation creates a double-copy overhead: tensors must first transfer from host memory to a temporary pinned buffer, then from this buffer to GPU memory. This two-stage process doubled transfer latency in our measurements—with a 20GB SDXL model requiring 8.7 seconds for data movement alone during cold starts.

The technical challenge extends beyond simply replacing pageable with pinned memory. Naively applying pinned memory creates three significant issues: (1) excessive host memory fragmentation due to non-contiguous allocation patterns, (2) system instability risks as pinned memory cannot be swapped, potentially causing out-of-memory conditions, and (3) increased allocation latency due to the complex physical memory operations required for pinning. These limitations are particularly severe in diffusion model serving where memory allocation patterns vary drastically across models (1-20GB base models) and request types (single vs. multi-image generation).

MLoRA implements a specialized heterogeneous pinned memory architecture with dynamic tier optimization:

$$\mathcal{M}_{alloc} = \{(s_i, f_i) | s_i \text{ is size}, f_i \text{ is frequency}\} \tag{7}$$

First, we continuously profile memory allocation patterns, collecting size-frequency distributions $\mathcal{M}_{alloc}$ across all inference requests. Using weighted K-means clustering with allocation size as the feature and frequency as the weight, we identify optimal memory size tiers:

$$\mathcal{M}_{clusters} = \text{WeightedKMeans}(\mathcal{M}_{alloc}, k = 5, w_i = f_i) \tag{8}$$

For each identified cluster centroid, we pre-allocate dedicated pinned memory pools with sizes strategically aligned to optimize both CUDA transfers and system memory management.

Size-based tier routing that maps allocation requests to the appropriate pre-pinned pool, minimizing wasted memory while ensuring efficient transfers:

$$\mathcal{A}_{best} = \arg \min_{\mathcal{P}_i \in \mathcal{P}, \text{size}(\mathcal{P}_i) \geq s_{req}} \text{size}(\mathcal{P}_i) - s_{req} \tag{9}$$

This approach routes each tensor allocation request of size s_{req} to the smallest suitable pre-pinned memory pool, reducing internal fragmentation while maintaining the performance benefits of pinned memory transfers.

6.3 Hierarchical Prefetching

6.3.1 Pipeline Combination Management. Our analysis of production workloads revealed that while the theoretical pipeline combination space exceeds 1.6 million configurations, actual usage follows a predictable pattern where certain combinations occur with high frequency. We leverage this insight to implement a hierarchical approach to pipeline combinations:

Static High-Frequency Combinations: For the top 50 base model + LoRA combinations (accounting for 73.2% of requests), we maintain persistent pairings with pre-computed fusion weights. This approach treats frequent combinations as atomic units rather than separate components, effectively *materializing* the computational graph ahead of request time. By precomputing weight fusion operations, we eliminate runtime tensor manipulations and matrix additions, reducing adapter loading and fusion overhead by 91.7%. This strategy recognizes that in highly skewed distributions, treating the head of the distribution as special cases yields disproportionate performance benefits at modest storage costs.

Dynamic Low-Frequency Combinations: For the remaining 26.8% of requests spanning thousands of unique configurations, we employ adaptive on-demand fusion with intelligent component indexing across the memory hierarchy. Our *asymmetric loading strategy* prioritizes base model availability—recognizing that base models define the dimensional constraints for adapter operations—while concurrently prefetching adapters along optimized I/O paths. We further exploit temporal locality through semantic cohesion caching, extending cache lifetimes for related components based on the empirical observation that users iteratively explore creative variants within short time windows.

Controlled Eviction: Traditional cache eviction policies such as LRU and LFU operate under the implicit assumption of component homogeneity, treating all cached objects as equivalent units differing only in access frequency or recency. However, in diffusion model serving, components exhibit fundamentally heterogeneous characteristics across multiple dimensions: loading costs vary by orders of magnitude (base models require 20-40s while small adapters load in <1s), memory footprints span from megabytes to gigabytes, and reuse patterns follow complex temporal and semantic correlations rather than simple recency patterns.

To address this heterogeneity, we develop a *multi-dimensional utility-based eviction strategy* that explicitly models the economic cost of each eviction decision. Our approach formulates eviction as an optimization problem where we seek to maximize the expected future utility of the cache configuration. The eviction score for c_i is computed as:

$$E_{score}(c_i) = \frac{P_{reuse}(c_i) \times T_{load}(c_i) \times \alpha_{type}(c_i)}{M_{size}(c_i) \times (1 + \beta \cdot \text{age}(c_i))} \quad (10)$$

where $P_{reuse}(c_i)$ represents the probability of near-term reuse estimated through a combination of temporal access patterns and semantic similarity clustering, $T_{load}(c_i)$ captures the complete loading latency including network transfer, decompression, and GPU initialization costs, $M_{size}(c_i)$ reflects the memory consumption, $\alpha_{type}(c_i)$ is a component-type weighting factor that accounts for the differential impact of evicting base models versus adapters, and $\beta \cdot \text{age}(c_i)$ introduces a temporal decay factor to prevent indefinite cache pollution.

This formulation embodies a fundamental insight: the value of retaining a component in cache is proportional to both the cost of reconstruction and the likelihood of future access, while being inversely related to the opportunity cost of the memory it occupies. Components with high reload costs and strong reuse signals should be preserved even under memory pressure, while large, infrequently accessed components become prime eviction candidates regardless of their individual loading costs.

7 Evaluation

7.1 Experimental Setup

We evaluated the effectiveness of MLoRA on a production cluster consisting of 300 GPUs. To ensure fair comparison, we employed an A/B testing methodology, dividing the cluster into two equivalent partitions [41]. Both environments processed the same production workload distribution, with requests randomly assigned to either group. The infrastructure was deployed on Kubernetes, operating in a serverless configuration to maximize operational flexibility [3, 67].

Baseline. Our baseline represents a production-grade ML serving infrastructure optimized over six months of continuous refinement [18, 70]. The system integrates state-of-the-art components: SGLang for high-performance inference, HuggingFace PEFT for LoRA integration, and DLoRA's online fusion strategy [23, 63]. The caching layer employs optimized LRU policies with multi-tier eviction strategies, while request routing uses enhanced round-robin load balancing with health-checking optimizations [19, 27]. This baseline already incorporates extensive performance enhancements from both academic research and industry practice, representing current state-of-the-art in diffusion model serving [6, 10]. Using this highly-optimized baseline ensures our evaluation reflects genuine architectural improvements rather than incremental optimizations.

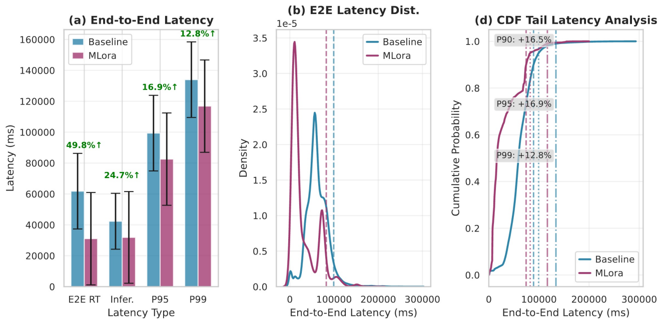


Figure 10: End-to-end performance comparison showing MLoRA achieves significant latency reduction compared to the baseline.

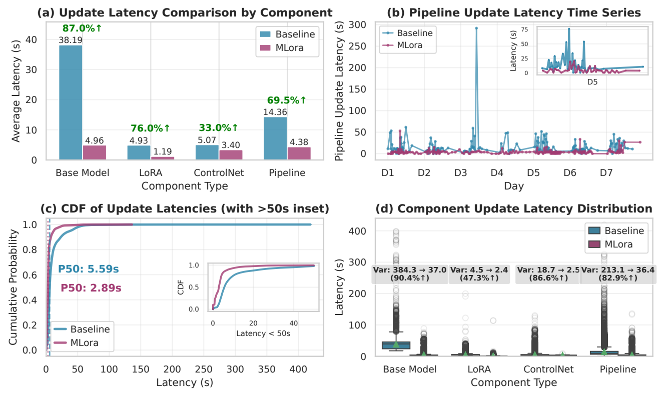


Figure 11: Model loading acceleration results across different component types. MLoRA shows significant latency reductions: 68.2% for base models (22.6s → 7.2s), 73.1% for LoRA adapters (4.6s → 1.2s), and 61.8% for ControlNet modules (5.1s → 1.9s).

7.2 End-to-End Performance Improvements

MLoRA achieves a 49.8% reduction in average end-to-end latency and 12.8% improvement in p99 tail latency (Fig. 10). These improvements stem from three synergistic optimizations targeting diffusion model serving bottlenecks.

Intelligent cache placement eliminates cold-start penalties through predictive component positioning across memory tiers. Heterogeneous memory optimization minimizes data movement by exploiting the full storage hierarchy during loading and inference. Cost-aware scheduling aligns requests with optimal server configurations based on computational capacity and component locality.

These mechanisms exhibit multiplicative effects: improved resource efficiency increases throughput, creating additional consolidation opportunities. This holistic approach proves critical for memory-intensive diffusion workloads where skewed access patterns and component heterogeneity defeat traditional optimization strategies.

7.3 Model Loading Acceleration

MLoRA achieves dramatic model loading acceleration across all component types (Fig. 11): 68.2% reduction for base models (22.6s → 7.2s), 73.1% for LoRA adapters (4.6s → 1.2s),

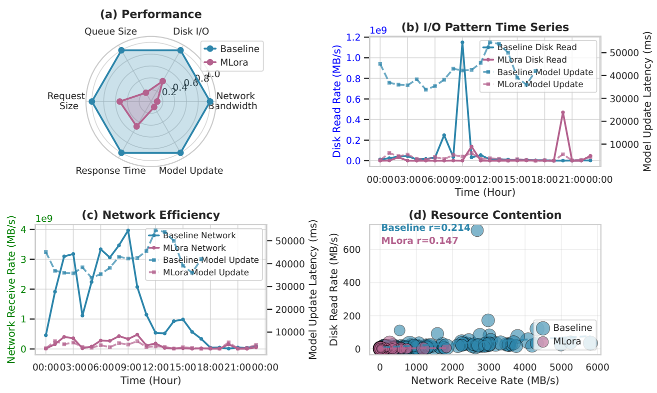


Figure 12: Network and I/O performance analysis comparing MLoRA with baseline. MLoRA achieves 89.5% reduction in network bandwidth and 60.2% reduction in disk I/O usage while significantly improving model update latency.

and 61.8% for ControlNet modules (5.1s → 1.9s). Complete pipeline updates—the end-to-end operation users experience—improve by 69.5% (14.4s → 4.4s).

This acceleration results from three synergistic optimizations: direct I/O with optimized block sizes bypassing kernel buffering, heterogeneous pinned memory management eliminating data path copies, and component-aware prefetching enabling parallel loading of interdependent elements. Together, these techniques address the fundamental I/O bottleneck where over 70% of cold-start latency originated from loading operations.

7.4 Network and I/O Performance

MLoRA fundamentally transforms diffusion model serving by addressing the core I/O bottleneck through hierarchical memory orchestration (Fig. 12a). By maintaining hot components in optimal memory tiers and intelligently evicting cold ones, we achieve 89.5% network bandwidth reduction and 60.2% disk I/O reduction—converting expensive remote transfers into efficient local memory operations.

The performance impact extends beyond raw bandwidth metrics to fundamental operational improvements. Model update latency decreases by 89.1% (from 38.9s to 4.2s) (Fig. 12b), queue size shrinks by 82.9% (Fig. 12c), and average response time improves by 51.4% (Fig. 12d). These gains demonstrate how addressing the underlying data movement inefficiencies creates cascading benefits throughout the entire serving stack. By optimizing the critical path from distributed storage to GPU memory, we transform what was previously an I/O-bound system into a more balanced architecture capable of efficiently utilizing computational resources. The reduced queue sizes further indicate that our approach successfully addresses the resource contention issues identified in our workload analysis, where popular model combinations created bottlenecks despite available system-wide capacity.

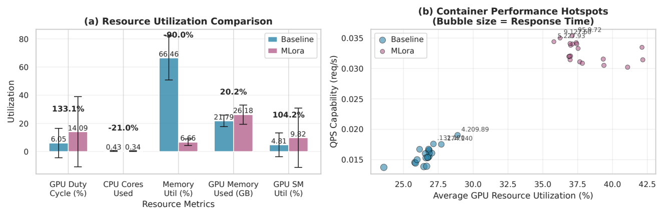


Figure 13: Container-level load distribution analysis. (a) Resource utilization comparison between baseline and MLoRA, showing significant improvements in GPU utilization while reducing memory overhead. (b) Performance-resource correlation analysis demonstrating how MLoRA transforms diffusion serving from an I/O-bound to a balanced system.

7.5 Container Load Distribution

Our container-level analysis reveals significant resource utilization improvements across the cluster. MLoRA achieves a 107.4% increase in per-container QPS while maintaining more balanced workload distribution. GPU duty cycle more than doubles from 6.0% to 14.1%, while GPU streaming multiprocessor utilization increases from 4.8% to 9.8% (Fig. 13a). This indicates more efficient computational resource usage despite the memory-bound nature of diffusion operations. Concurrently, host memory utilization decreases dramatically from 66.5% to 6.7%, demonstrating how our heterogeneous instance design with specialized memory management eliminates redundant buffer allocations that plagued the baseline system.

The performance-resource relationship analysis (Fig. 13b) highlights how MLoRA fundamentally alters system behavior—containers achieve both higher throughput and greater resource efficiency, creating a positive correlation between utilization and performance that contrasts with the negative correlation observed in conventional architectures. This efficiency stems from three key optimizations: the global cache coordination that reduces redundant model loading across instances, the heterogeneous fast I/O paths that accelerate data movement operations, and the cost-aware scheduling that ensures optimal request placement. Collectively, these improvements transform diffusion model serving from an I/O-bound workload where resources remain underutilized despite high user-perceived latency into a balanced system that efficiently utilizes available resources while delivering superior performance.

8 Related Work

ML Serving Systems. Traditional machine learning serving systems [19, 31, 35, 43, 46, 57, 65, 68, 70, 76] like Alpaserve [35] and vLLM [31] optimize inference for stable model configurations with predictable access patterns. These systems address model batching and resource allocation but assume workloads fundamentally different from diffusion models' combinatorial component spaces and extreme skewness.

LoRA and Adapter Serving. The emergence of LoRA adapters has motivated specialized serving systems [9, 54, 63] like Punica [9], and dLoRA [63]. These systems focus primarily on efficient GPU kernel design for adapter fusion and memory management during inference, but do not address the broader system-level challenges of cache hierarchy management, request skewness, and multi-component pipeline coordination that emerge in production diffusion serving environments.

GPU Cluster Management. GPU cluster scheduling systems [17, 25, 26, 34, 48, 55, 56, 61, 62] like Llumnix [61], Pollux [48], and AntMan [66] optimize resource allocation for training workloads with different performance characteristics than inference serving. While these systems address GPU memory management and scheduling efficiency, they target batch training jobs rather than the latency-sensitive, memory-bound characteristics of diffusion model serving with its complex multi-tier caching requirements.

Distributed Caching Systems. Prior work on distributed caching [19, 22, 27, 32, 53, 74] addresses general-purpose workloads with relatively uniform object sizes and access patterns. While systems like NetCache [27] optimize network-level caching and Cocktail [19] addresses multi-dimensional optimization, they do not address the combinatorial complexity and extreme size heterogeneity characteristic of diffusion model components.

9 Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback and suggestions that helped improve the quality of this paper. This work is supported by the National Natural Science Foundation of China (No. 92267105), Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515130002), Guangdong Special Support Plan (No. 2021TQ06X990), Shenzhen Basic Research Program (No. JCYJ20220818101610023, KJZD20230923113800001), and Alibaba Innovative Research (AIR) Project.

10 Conclusion

This paper presents a comprehensive analysis of diffusion model serving in production environments, revealing unique challenges that arise from their distinctive workload characteristics and resource requirements. Our analysis of over 3.5 million production requests demonstrates how extreme skewness in model popularity, combinatorial pipeline configurations, and high data movement costs create fundamental inefficiencies in conventional serving architectures. We developed MLoRA, a specialized serving system that implements global cache coordination, heterogeneous fast instances with optimized I/O, and hierarchical prefetching with cost-aware routing.

References

[1] Mart'in Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, et al. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://arxiv.org/abs/1603.04467v2.

[2] Rahaf Abdullah, Huiyang Zhou, and Amro Awad. 2023. Plutus: Bandwidth-Efficient Memory Security for GPUs. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023.* 543–555.

[3] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications.. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020.* 419–434.

[4] Shubham Agarwal, Subrata Mitra, Sarthak Chakraborty, Srikrishna Karanam, Koyel Mukherjee, and Shiv Kumar Saini. 2024. Approximate Caching for Efficiently Serving Text-to-Image Diffusion Models.. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024.*

[5] Abutalib Aghayev, Sage Weil, Michael Kuchnik, Mark Nelson, Gregory R. Ganger, and George Amvrosiadis. 2019. File Systems Unfit as Distributed Storage Backends: Lessons from 10 Years of Ceph Evolution. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville Ontario Canada, 353–369.

[6] Sohaib Ahmad, Hui Guan, Brian D. Friedman, Thomas Williams, Ramesh K. Sitaraman, and Thomas Woo. 2024. Proteus: A High-Throughput Inference-Serving System with Accuracy Scaling. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. La Jolla CA USA, 318–334.

[7] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing.. In *Proceedings of the 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018.* 923–935.

[8] Xiaohu Chai, Tianyu Zhou, Keyang Hu, Jianfeng Tan, Tiwei Bie, Anqi Shen, Dawei Shen, Qi Xing, et al. 2025. Fork in the Road: Reflections and Optimizations for Cold Start Latency in Production Serverless Systems. In *19th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2025, Boston, MA, USA, July 7-9, 2025.* 199–218.

[9] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023. Punica: Multi-Tenant LoRA Serving. arXiv:2310.18547 [cs]

[10] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing.. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022.* 199–216.

[11] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. Shanghai China, 153–167.

[12] Christina Delimitrou and Christos Kozyrakis. 2013. QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon. *ACM Transactions on Computer Systems* (Dec. 2013), 1–34.

[13] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. Lausanne Switzerland, 467–481.

[14] Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. 2024. MuxServe: Flexible Spatial-Temporal Multiplexing for Multiple LLM Serving. arXiv:2404.02015 [cs]

[15] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, et al. 2021. DAPPLE: A Pipelined Data Parallel Approach for Training Large Models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. Virtual Event Republic of Korea, 431–445.

[16] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024.* 135–153.

[17] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning.. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019.* USA, 485–500.

[18] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up.. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.* 443–462.

[19] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2022. Cocktail: A Multidimensional Optimization for Model Serving in Cloud.. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022.* 1041–1057.

[20] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-Scale Preemption for Concurrent GPU-accelerated DNN Inferences.. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022.* 539–558.

[21] Jaehoon Heo, Adiwena Putra, Jieon Yoon, Sungwoong Yune, Hangyeol Lee, Ji-Hoon Kim, and Joo-Young Kim. 2025. EXION: Exploiting Inter- and Intra-Iteration Output Sparsity for Diffusion Models. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025.* 324–337.

[22] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, et al. 2024. MemServe: Context Caching for Disaggregated LLM Serving with Elastic Memory Pool. arXiv:2406.17565 [cs]

[23] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs]

[24] Junhao Hu, Jiang Xu, Zhixia Liu, Yulong He, Yuetao Chen, Hao Xu, Jiang Liu, Jie Meng, et al. 2025. DEEPSERVE: Serverless Large Language Model Serving at Scale. In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025.* 57–72.

[25] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, et al. 2024. Characterization of Large Language Model Development in the Datacenter.. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024.* 709–729.

[26] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. 2023. Sia: Heterogeneity-aware,

Goodput-Optimized ML-cluster Scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 642–657.

[27] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soul'e, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*. Shanghai China, 121–136.

[28] Sungbin Kim, Hyunwuk Lee, Wonho Cho, Mincheol Park, and Won Woo Ro. 2025. Ditto: Accelerating Diffusion Model via Temporal Value Similarity. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025*. 338–352.

[29] Weihao Kong, Yifan Hao, Qi Guo, Yongwei Zhao, Xinkai Song, Xiaqing Li, Mo Zou, Zidong Du, et al. 2024. Cambricon-D: Full-Network Differential Acceleration for Diffusion Models. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. 903–914.

[30] Joyjit Kundu, Wenzhe Guo, Ali BanaGozar, Udari De Alwis, Sourav Sengupta, Puneet Gupta, and Arindam Mallik. 2024. Performance Modeling and Workload Analysis of Distributed Large Language Model Training and Inference. arXiv:2407.14645

[31] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, et al. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 611–626.

[32] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. 155–172.

[33] Jie Li, Laiping Zhao, Yanan Yang, Kunlin Zhan, and Keqiu Li. 2022. Tetris: Memory-efficient Serverless Inference through Tensor Sharing.. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*.

[34] Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, et al. 2024. CaraServe: CPU-Assisted and Rank-Aware LoRA Serving for Generative LLM Inference. arXiv:2401.11240 [cs]

[35] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, et al. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving.. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*. 663–679.

[36] Yanying Lin, Shijie Peng, Chengzhi Lu, Chengzhong Xu, and Kejiang Ye. 2025. FlexPipe: Adapting Dynamic LLM Serving Through Inflight Pipeline Refactoring in Fragmented Serverless Clusters. arXiv:2510.11938 [cs]

[37] Chengzhi Lu, Huanle Xu, Kejiang Ye, Guoyao Xu, Liping Zhang, Guodong Yang, and Chengzhong Xu. 2023. Understanding and Optimizing Workloads for Unified Resource Management in Large Cloud Platforms. In *Proceedings of the Eighteenth European Conference on Computer Systems*. Rome Italy, 416–432.

[38] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, et al. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*. 412–426.

[39] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling.. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*. USA, 289–304.

[40] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-Aware Performance Prediction For Virtualized Network Functions. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Virtual Event USA, 270–282.

[41] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, Robert Ricci, and Ana Klimovic. 2018. Taming Performance Variability.. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*. 409–425.

[42] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, et al. 2020. MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance. *IEEE Micro* (March 2020), 8–16.

[43] 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*. La Jolla CA USA, 1112–1127.

[44] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville Ontario Canada, 1–15.

[45] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, 'I nigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. 118–132.

[46] Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2024. Queue Management for SLO-Oriented Large Language Model Serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. New York, NY, USA, 18–35.

[47] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low Latency Geo-distributed Data Analytics. *ACM SIGCOMM Computer Communication Review* (Sept. 2015), 421–434.

[48] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning.. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*.

[49] Francisco Romero, Gohar Irfan Chaudhry, 'I nigo Goiri, Pragna Gopa, Paul Batum, Neeraja J. Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, et al. 2021. FaaT: A Transparent Auto-Scaling Cache for Serverless Applications. In *Proceedings of the ACM Symposium on Cloud Computing*. Seattle WA USA, 122–137.

[50] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: Warming Serverless Functions Better with Heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Lausanne Switzerland, 753–767.

[51] Keshav Santhanam, Siddharth Krishna, Ryota Tomioka, Andrew Fitzgibbon, and Tim Harris. 2021. DistIR: An Intermediate Representation for Optimizing Distributed Neural Networks. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. Online United Kingdom, 15–23.

[52] Mohammad Shahrad, Rodrigo Fonseca, 'I nigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, et al. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.. In *Proceedings of the 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*. 205–218.

[53] Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, Yuxin Su, Jiazhen Gu, Hao Feng, Yangfan Zhou, and Michael R. Lyu. 2023. Ditto: An Elastic and Adaptive Memory-Disaggregated Caching System. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 675–691.

[54] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, et al. 2024. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. arXiv:2311.03285 [cs]

[55] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E. Gonzalez, and Ion Stoica. 2024. Fairness in Serving Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. 965–988.

[56] Sudipta Saha Shubha, Haiying Shen, and Anand Iyer. 2024. USHER: Holistic Interference Avoidance for Resource Optimized ML Inference. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. 947–964.

[57] Chijun Sima, Yao Fu, Man-Kit Sit, Liyi Guo, Xuri Gong, Feng Lin, Junyu Wu, Yongsheng Li, et al. 2022. Ekko: A Large-Scale Deep Learning Recommender System with Low-Latency Model Upyear.. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*. 821–839.

[58] Xiaoniu Song, Yiwen Zhang, Rong Chen, and Haibo Chen. 2023. UGACHE: A Unified GPU Cache for Embedding-based Deep Learning. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 627–641.

[59] Ion Stoica and Scott Shenker. 2021. From Cloud Computing to Sky Computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. Ann Arbor Michigan, 26–32.

[60] Jovan Stojkovic, Chaojie Zhang, 'I nigo Goiri, Josep Torrellas, and Esha Choukse. 2025. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025*. 1348–1362.

[61] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. 173–191.

[62] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, et al. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters.. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*. 945–960.

[63] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*. 911–927.

[64] Shuaipeng Wu, Yanying Lin, Shijie Peng, Wenyan Chen, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, et al. 2025. Rock: Serving Multimodal Models in Cloud with Heterogeneous-Aware Resource Orchestration for Thousands of LoRA Adapters. In *2025 IEEE International Conference on Cluster Computing (CLUSTER)*. 1–13.

[65] Haojun Xia, Zhen Zheng, Xiaoxia Wu, Shiyang Chen, Zhewei Yao, Stephen Youn, Arash Bakhtiari, Michael Wyatt, et al. 2024. Quant-LLM: Accelerating the Serving of Large Language Models via FP6-Centric Algorithm-System Co-Design on Modern GPUs. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*. 699–713.

[66] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, et al. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning.. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. 533–548.

[67] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. 2022. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Lausanne Switzerland, 768–781.

[68] 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 2022, Carlsbad, CA, USA, July 11-13, 2022*. 521–538.

[69] Shaoxun Zeng, Minhui Xie, Shiwei Gao, Youmin Chen, and Youyou Lu. 2025. Medusa: Accelerating Serverless LLM Inference with Materialization. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*. 653–668.

[70] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the Wild.. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. 787–808.

[71] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding Conditional Control to Text-to-Image Diffusion Models. arXiv:2302.05543 [cs]

[72] Yuxuan Zhang and Sebastian Angel. 2025. Quilt: Resource-aware Merging of Serverless Workflows. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSP 2025, Lotte Hotel World, Seoul, Republic of Korea, October 13-16, 2025*. 907–927.

[73] Yanqi Zhang, 'I nigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. Virtual Event Germany, 724–739.

[74] Youpeng Zhao, Di Wu, and Jun Wang. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. 1005–1017.

[75] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, et al. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning.. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*. 559–578.

[76] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, et al. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs]