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

Yanying Lin Shuaipeng Wu Shutian Luo Hong Xu
Haiying Shen Chong Ma Min Shen Le Chen
Chengzhong Xu Lin Qu Kejiang Ye

SIAT; UCAS; SUSTech; UVA; CUHK; UMacau; Alibaba

November 21 2025, SoCC

Outline

Diffusion Pipeline Architecture

- Base models (1–20 GB): SDXL, SD1.5, SD3, FLUX
- LoRA adapters (100 MB–1 GB): style customization
- ControlNet modules (0.5–10 GB): conditional generation
- Components exhibit distinct lifetimes and resource footprints

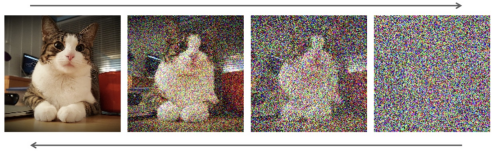

Figure: Stable Diffusion image generation process.

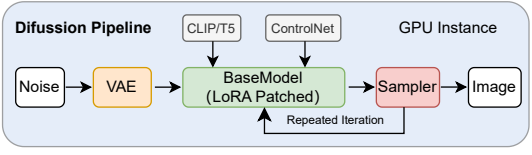

Figure: Diffusion pipeline DAG structure.

Why Stable Diffusion Serving?

- **Stable Diffusion (SD)** and LLM Serving are the two dominant inference workloads in production clusters
- Both reach 10K+ QPS at peak, but SD generation takes tens of seconds vs. LLM's token-level latency
- SD operates through multi-stage pipelines with base models, LoRA adapters, and ControlNet modules
- Over 1.6 million possible pipeline combinations in production environments create unprecedented system complexity

SD vs. LLM: Core Differences

Stable Diffusion:

- Multi-stage, iterative denoising pipeline
- Convolution-based computation
- Memory bandwidth and capacity critical
- Activation memory is the bottleneck

Large Language Models:

- Autoregressive, single-stage engine
- Matrix multiplication (GEMM) dominant
- Compute throughput and latency sensitive
- Model weights and KV Cache are bottlenecks

SD is a memory-capacity/bandwidth-sensitive image processing pipeline, while LLM is a compute-throughput/communication-sensitive sequence generation engine.

Production Challenges

- Daytime workload spikes drive hot-model contention
- Long-tail requests trigger expensive cold loads
- Multi-image jobs (up to 8 outputs) consume full 24 GB GPUs
- ControlNet and adapter diversity prevent effective batching

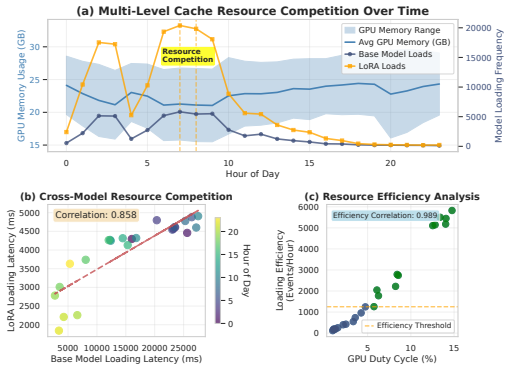

Figure: Multi-level resource contention.

Insight 1: Request Skew Causes Performance Inversion

University of Chinese Academy of Sciences

Key Finding: Model Popularity Skew

- Gini coefficient: 0.876 (extreme inequality)
- Top 5% models handle 78% of all requests
- Paradox: Hot models complete in 29 s vs. 25.8 s for cold ones
- Resource contention offsets cache warmth benefits

SD models exhibit unprecedented request concentration, yet traditional caching strategies fail due to contention.

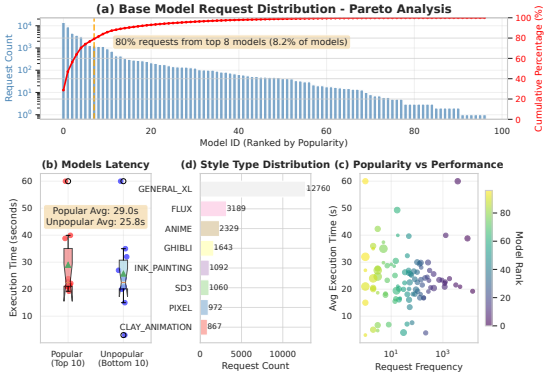

Figure: Base model request distribution and latency paradox.

Insight 1: Implications

- Head models need persistent multi-replica placement with GPU reservation
- Tail models should rely on host/SSD tiers with prefetch triggers
- Schedulers must route by popularity bands to separate hot and cold pools

Insight 2: Resource Utilization Paradox

Key Finding: GPU Underutilization Despite High Latency

- 98.4% of pods operate below 20% GPU utilization
- Yet users experience high latency
- Computation consumes only 15–30% of end-to-end latency
- 70–85% of time lost to orchestration overhead

SD serving is memory-bandwidth bound, not compute-bound. Traditional GPU utilization metrics are misleading.

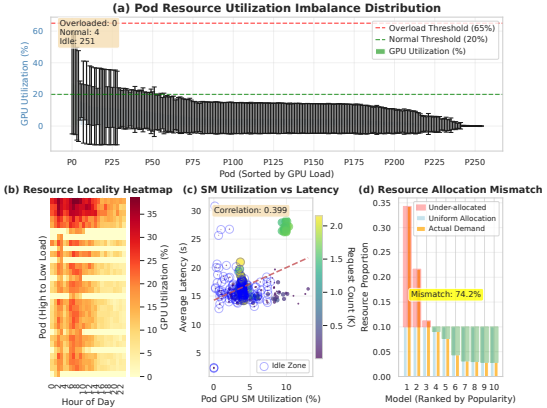

Figure: Resource allocation mismatch.

Insight 2: Implications

- Memory bandwidth, not SM compute, caps throughput
- Scheduler decisions must be memory-aware, not compute-centric
- Utilization metrics must expose memory bandwidth saturation

Insight 3: Cache Replacement Overhead

Key Finding: Asymmetric Replacement Costs

- Base model refreshes cost 22.6 s (5× LoRA updates)
- Replacement bursts: network traffic +345%, disk I/O only +26%
- Single eviction can erase minutes of warm state

SD's heterogeneous components require cost-aware eviction policies. Simple LRU fails catastrophically.

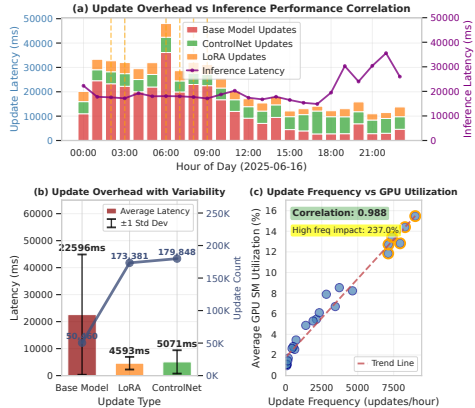

Figure: Component-specific replacement costs.

Insight 3: Implications

- Cost-aware eviction policies must weight load cost, reuse probability, and size
- Replacement must coordinate with scaling to avoid cache thrashing
- Simple LRU fails for heterogeneous component characteristics

MLoRA Architecture Overview

- **Global Cache Coordination:** Unified view across storage tiers, popularity-aware distribution
- **Heterogeneous Instance Manager:** Fast I/O paths and pinned-memory templates
- **Hierarchical Request Router:** Cost-aware scheduling with cache locality

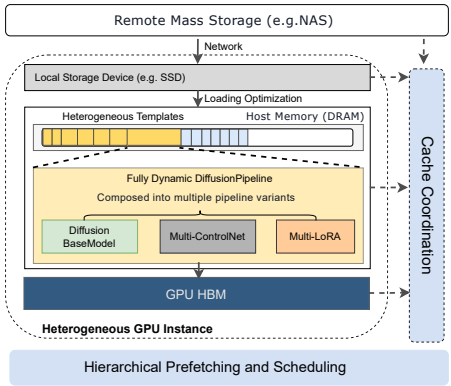

Figure: Integrated system architecture.

Technique 1: Global Cache Coupled with Scaling

- Treat entire diffusion pipelines as atomic caching units
- Three-tier hierarchy: remote storage $\rightarrow$ SSD $\rightarrow$ host memory $\rightarrow$ GPU memory
- **Hot models (top 5%):** Multi-replica persistence with pre-computed fusion weights
- **Warm models (5–30%):** Multi-armed bandit temporal prefetching
- **Cold models (bottom 70%):** Time-decayed opportunistic caching
- Cache-aware scaling decisions reduce scaling latency by 62.4%

Technique 2: Heterogeneous Fast Instances

- **Direct I/O:** 2 MB blocks reduce system call overhead by 16×
- **Heterogeneous pinned memory:** Weighted k-means clustering identifies 5 memory tiers
- Pre-allocated pinned memory pools eliminate double-copy overhead
- Scaling delay reduced from 40–60 s to seconds
- Model loading time reduced by over 70%

Technique 3: Hierarchical Prefetching

- **Static high-frequency combinations:** Top 50 pairs (73.2% requests) with pre-computed fusion weights reduce overhead by 91.7%
- **Dynamic low-frequency:** Asymmetric loading prioritizes base models, concurrently prefetches adapters
- **Multi-dimensional utility-based eviction:** Considers reuse probability, load cost, component type, and memory footprint
- Replaces traditional LRU with cost-aware policies

End-to-End Performance

- Average latency: 49.8% reduction
- p99 latency: 12.8% improvement
- Queue sizes: 82.9% reduction
- Workload-aware placement improves responsiveness

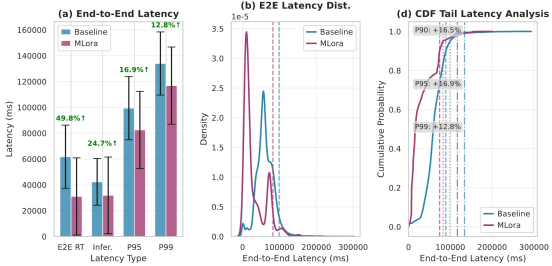

Figure: Latency distribution comparison.

Model Loading Acceleration

- Base models: 22.6 s → 7.2 s (68.2% reduction)
- LoRA adapters: 4.6 s → 1.2 s (73.1% reduction)
- ControlNet: 5.1 s → 1.9 s (61.8% reduction)
- Full pipeline: 14.4 s → 4.4 s (69.5% reduction)

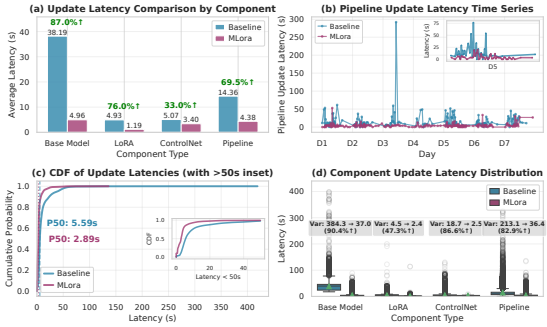

Figure: Component-level loading improvements.

Resource Efficiency

- Network bandwidth: 89.5% reduction
- Disk I/O: 60.2% reduction
- GPU duty cycle: 6.0% → 14.1%
- SM utilization: 4.8% → 9.8%

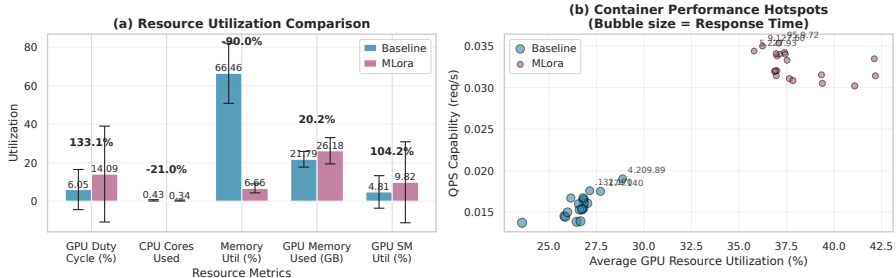

Figure: Container-level resource improvements.

Production Dataset

- 3.5 million requests across 300+ GPUs
- Complete execution stack instrumentation:
  - Application-level: request patterns, model variants
  - Middleware-level: loading times, pipeline DAG execution
  - Hardware-level: GPU utilization, memory, I/O
- First holistic view of diffusion serving in production

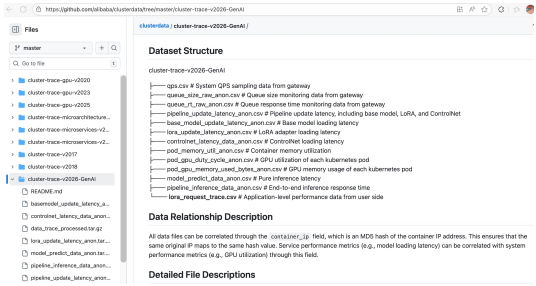

Figure: Dataset overview and statistics.

Dataset available at: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2026-GenAI

Questions?

For more information, Yanying —yy.lin1@siat.ac.cn

Homepage: https://yanyinglin.github.io

Dataset available at: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2026-GenAI