GENSERVE: Efficient Co-Serving of Heterogeneous Diffusion Model Workloads

Step-Level Resource Adaptation for Mixed Text-to-Image and Text-to-Video Serving on Shared GPU Clusters

Fanjiang Ye, Zhangke Li, Xinrui Zhong, Ethan Ma, Russell Chen, Kaijian Wang, Jingwei Zuo, Desen Sun,
Ye Cao, Triston Cao, Myungjin Lee, Arvind Krishnamurthy, Yuke Wang

arXiv:2604.04335

Presented by Fanjiang Ye

April 28

Contents

OVERVIEW

01

Background & Motivation

The Diffusion Inference Process

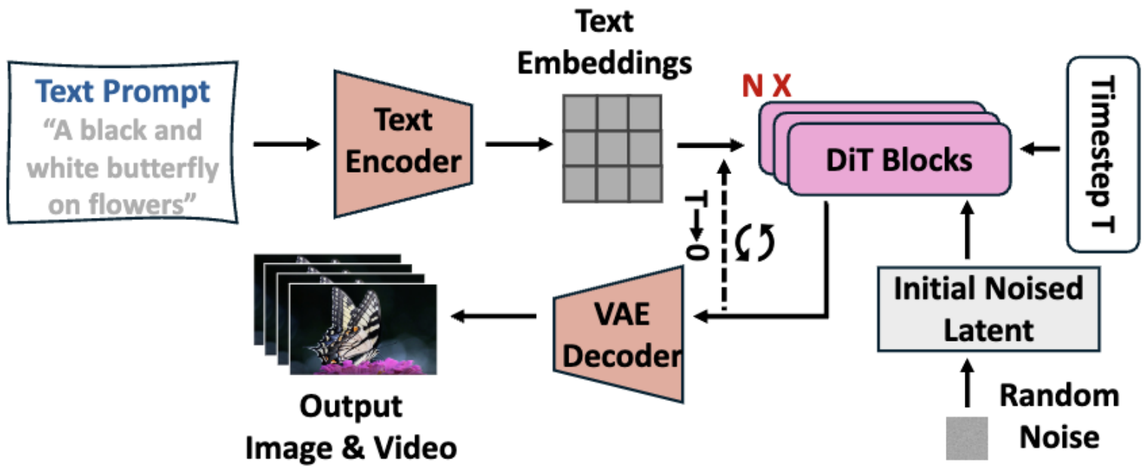


Fig. 1: Overview of DiT inference process.

Architecture:

VAE: RGB <-> Latent Space

Text Encoder

DiT

Why Co-serving Attractive (1)

Diffusion models weights are much smaller than LLM. (no need to do Model Parallelism)

Peak memory footprint is **unchanged**[1] in denoising, unlike AR. (Can fit multiple models in single GPU, total ~50 GB).
Constant KV Cache length.

Image generation models (primarily text-to-image)

Model	Params (public)	FP16 weights (approx.)	Notes
Stable Diffusion 1.5	0.983B	~2.0 GB	
SDXL 1.0 (base)	3.5B	~7.0 GB	
SDXL UNet (backbone only)	2.6B	~5.2 GB	UNet param count often quoted separately
Stable Diffusion 3 Medium	2B	~4.0 GB	
Stable Diffusion 3.5 Medium	2.5B	~5.0 GB	
PixArt-Σ	0.6B	~1.2 GB	
PixArt-Sigma	0.7B	~1.4 GB	
Stable Cascade	5.1B	~10.2 GB	
FLUX.1-dev	12B	~24 GB	
Qwen-Image	20B	~40 GB	
Hunyuan-DiT-S	0.7B	~1.4 GB	Repo notes also mention VRAM ranges

Video generation models (primarily text-to-video / image-to-video)

Model	Params (public)	FP16 weights (approx.)	Notes
CogVideoX / CogVideoX-1.5 (5B tier)	5B	~10 GB	CogVideoX is commonly referenced as 2B/5B tiers; 5B is the typical "large" tier
HunyuanVideo-1.5	8.3B	~16.6 GB	
Wan2.1 (14B)	14B	~28 GB	
Wan2.1 (1.3B)	1.3B	~2.6 GB	
Open-Sora 1.2	1.1B	~2.2 GB	
VideoCrafter2 (commonly cited)	~1.4B	~2.8 GB	

1. When the input (resolutions, # frames) are fixed.

Why Co-serving Attractive (2)

Replicated Partitioning is **better than** Dedicated Partitioning.

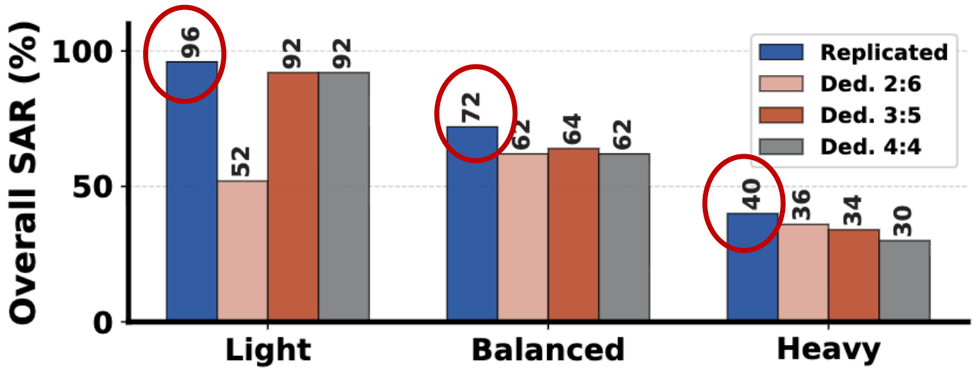


Figure 15: Dedicated GPU partitioning (X:Y means X image GPUs, Y video GPUs) vs. replicated co-serving across workload mixes. The replicated design leads in all mixes by avoiding resource fragmentation.

Challenges in Co-Serving Diffusion Tasks (1)

Workload Asymmetry

Table 3: Per-step arithmetic intensity of DiT for T2I (SD3.5) and T2V (Wan2.2-5B, 81 frames) in BF16.

Model	Res.	Seq. Len.	FLOPs/step (T)	AI (FLOPs/B)
SD3.5 (T2I)	256p	256	0.36	243
	480p	900	1.34	764
	720p	2,304	3.91	1,646
Wan2.2 (T2V)	256p	1,344	10.81	1,197
	480p	4,725	43.90	3,437
	720p	12,096	145.26	6,941

up to 20x per-step cost differences.

Very Different SLOs.

Challenges in Co-Serving Diffusion Tasks (2)

Conventional Scheduling Bottlenecks

FCFS (First-come-first-serve): Head-of-line Blocking, image SLO drops.

SJF: Starving video requests, video SLO drops.

No policy can balance both modalities

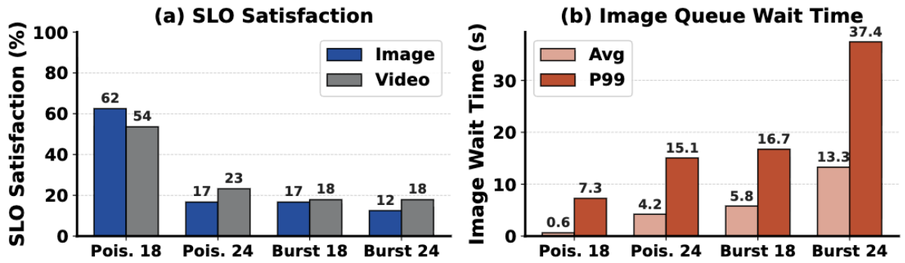


Figure 4: Head-of-line blocking under FCFS scheduling with workloads (70% video, 81 frames). Here Pois. means the Poisson arrival pattern. (a) SLO satisfaction drops sharply under bursty video arrivals, image SLO falls from 62% to 12% as videos monopolize all GPUs. (b) Image P99 queue wait time increases by 5× from 7.3 s to 37.4 s, confirming that FCFS cannot protect image requests, causing HOL blocking.

Challenges in Co-Serving Diffusion Tasks (3)

Static Resource Mismatch

Existing systems use a single fixed config tuned for one task type.
E.g., fixed batch size and sequence parallelism degree.

However, for co-serving diffusion tasks,
optimal configuration varies continuously across workload mix, resolutions and device state.

Key Insights

Predictable & Preemptible

Stable per-step running cost (correlation value < 0.05%). Unlike AR LLM with unpredictable output length, almost fully predictable.

Naturally preemptible at iterative step boundaries.
Negligible overhead of pause and resume (< 0.15% of step time)

Table 1: Per-step DiT runtime stability across batch sizes and SP degrees. Std and CV (Coefficient of Variation) are measured over repeated 50-step runs.

Image (SD 3.5)				Video (Wan2.2-5B)			
BS	SP	Std (ms)	CV (%)	BS	SP	Std (ms)	CV (%)
1	1	0.03	0.04	1	1	0.12	0.02
2	1	0.04	0.03	1	2	0.10	0.01
4	1	0.05	0.02	1	4	0.16	0.02
8	1	0.07	0.01	1	8	0.14	0.01

Table 7: Preemption overhead summarized by 720p video with different sequence parallelism (SP) degrees. Results are averaged across 5 independent runs.

Res.	SP	Base Step (ms)	Pause (μs)	Resume (ms)	Resume/Step (%)
720p	1	781.57	3.40	0.036	0.005
	2	780.16	3.05	0.464	0.060
	4	780.66	3.00	0.592	0.076
	8	781.15	3.05	0.868	0.112

Key Insights

Different Compute Profiles (1)

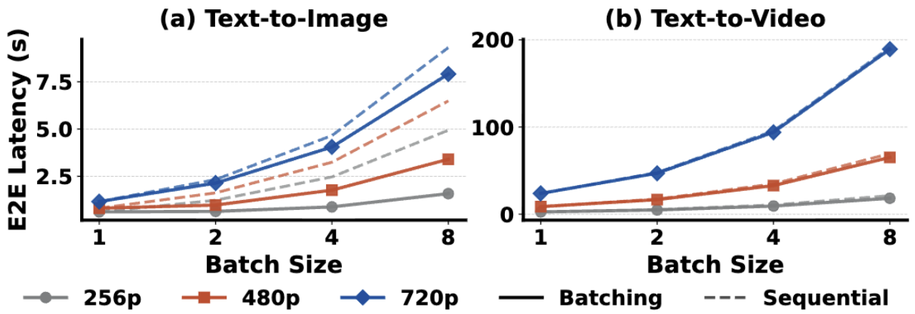


Figure 3: End-to-end latency of T2I and T2V workloads across batch sizes and resolutions. In T2I, batching yields noticeable savings over theoretical sequential execution at low resolutions, while T2V exhibits limited room for batching-based latency reduction even for low-resolution requests.

Batch Size:

T2I is compute-underutilized at low resolutions. Batching improves throughput.

T2V is compute-intensive even at low resolutions. Limited batching gains.

Key Insights

Different Compute Profiles (2)

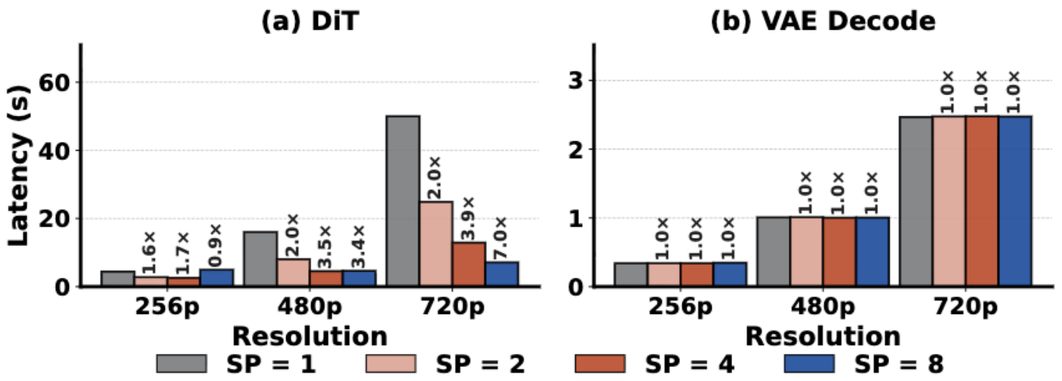


Figure 5: The runtime of different stages in T2V across different resolutions and Sequence Parallelism (SP) degrees. DiT and VAE Decode latency of T2V across resolutions and SP degrees. DiT benefits from higher SP at high resolutions (up to 7.0× at 720p/81f) but shows diminishing returns at low resolutions. VAE Decode latency is unaffected by SP degree.

Sequence Parallelism Degree:

For DiT: higher SP degrees for larger resolutions. Why?

For VAE: Keep on a single GPU due to low computation workload.

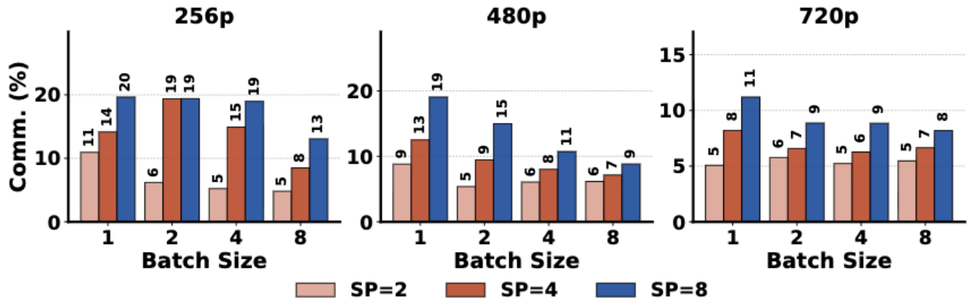


Figure 6: Communication overhead (as % of per-step time) for T2V DiT across resolutions, batch sizes, and SP degrees. Higher SP degrees incur more communication; at low resolutions (256p) the ratio reaches 20%, explaining the poor SP scalability in Figure 5. Larger batch sizes and higher resolutions reduce the ratio by increasing per-GPU computation.

Key Insights

Lightweight Online Scheduler & Joint Optimization

Static deployment can not track runtime workload fluctuations.
Over-provisioning one modality will starve the other. (Fig. 15)

MILP solvers take minutes, far too slow for millisecond-scale scheduling.

Preemption, parallelism, batching are individually beneficial, need a joint coordination.

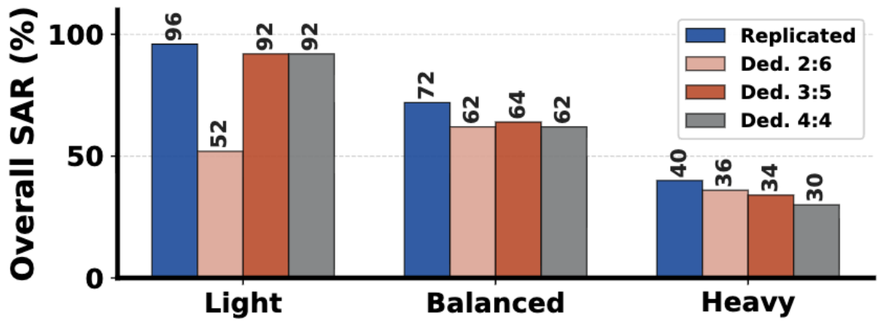


Figure 15: Dedicated GPU partitioning (X:Y means X image GPUs, Y video GPUs) vs. replicated co-serving across workload mixes. The replicated design leads in all mixes by avoiding resource fragmentation.

Opportunity:

Formulate a lightweight online SLO-aware scheduler.

Jointly decide GPU allocations at each step round.

02

GENSERVE Design

System Overview

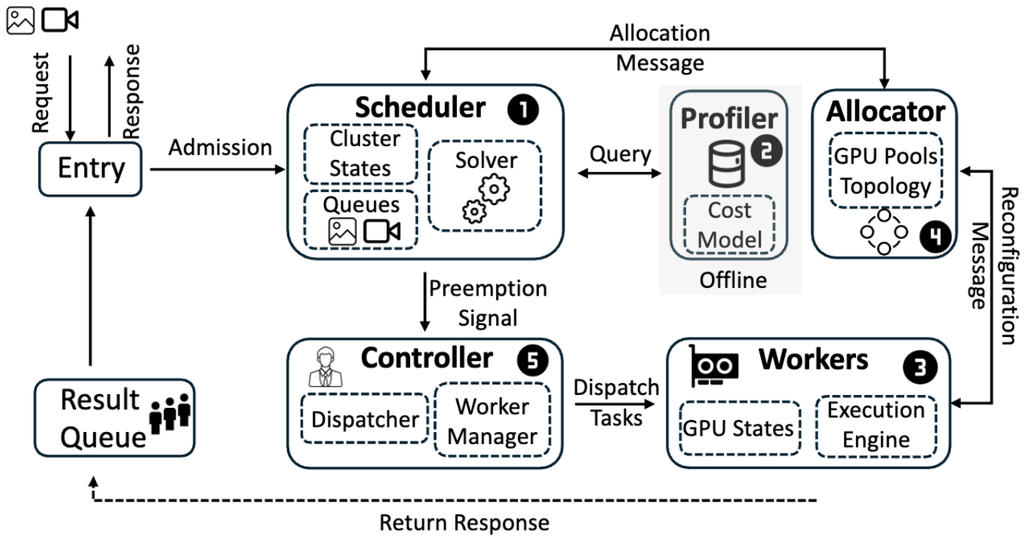


Figure 7: System overview of GENSERVE. Incoming requests are admitted by the Scheduler, which maintains cluster states and queries the Profiler for offline latency estimates to invoke the Solver for joint optimization. The resulting decisions are passed to the Allocator and the Controller, which dispatches tasks and preemption signals to Workers. Workers execute inference on assigned GPUs and report step-level progress back through the Result Queue.

Request Life Cycle

Request arrives -> **Scheduler** record task type, resolution, deadline.

Scheduler queries **Profiler** for offline latency estimates across resolutions, batch sizes, and SP degrees.

Solver constructs global snapshot and solves joint optimization over all in-flight requests.

Allocator translates decisions into GPU assignments and adjusts parallelism topology.

Controller delivers tasks and preemption signals to **Workers**.

Intelligent Video Preemption (1)

Potential Candidate Selection

Deadline Slack: $\text{slack}_v = D_v - t_{\text{now}} - S_v^{\text{rem}} \cdot T_{\text{step}}(v)$.

Scheduler ranks them by **descending slack** and selects highest-slack victims.

Pause: full preemption or downgrading SP degrees (Softer way).

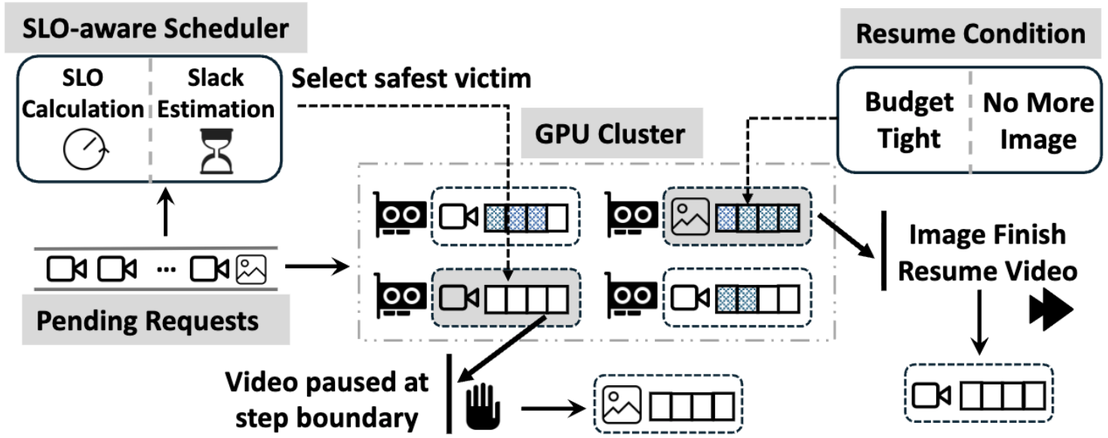


Figure 8: Illustration of intelligent video preemption. The scheduler computes the slack of running video jobs, pauses the safest one at a step boundary to serve an urgent image request, and resumes it when its deadline budget becomes tight, or no more image requests arrive.

Intelligent Video Preemption (2)

Deadline-aware Resume Policy

Trigger resumption when:

1) budget-tight: the remaining time falls below estimated completion.

2) GPU is idle: no new images arrive.

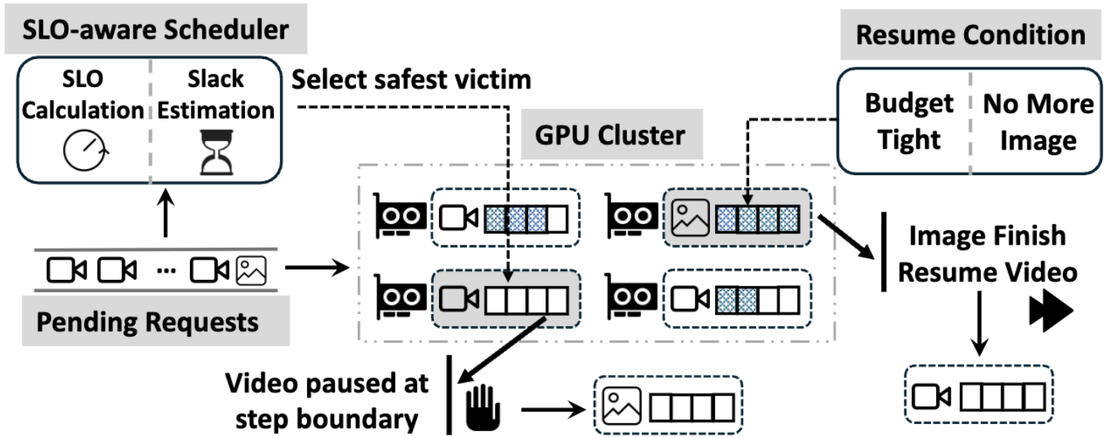


Figure 8: Illustration of intelligent video preemption. The scheduler computes the slack of running video jobs, pauses the safest one at a step boundary to serve an urgent image request, and resumes it when its deadline budget becomes tight, or no more image requests arrive.

Adaptive Resource Allocation (1)

Elastic Sequence Parallelism (T2V)

Stage Decoupling: DiT (95% of time) is parallelized via SP across GPUs.
VAE decode always runs on a single GPU.

Runtime SP Switching:
1) When images arrives and GPUs are scarce: downgrade SP degree.
2) When GPUs become idle: upgrade SP degree.

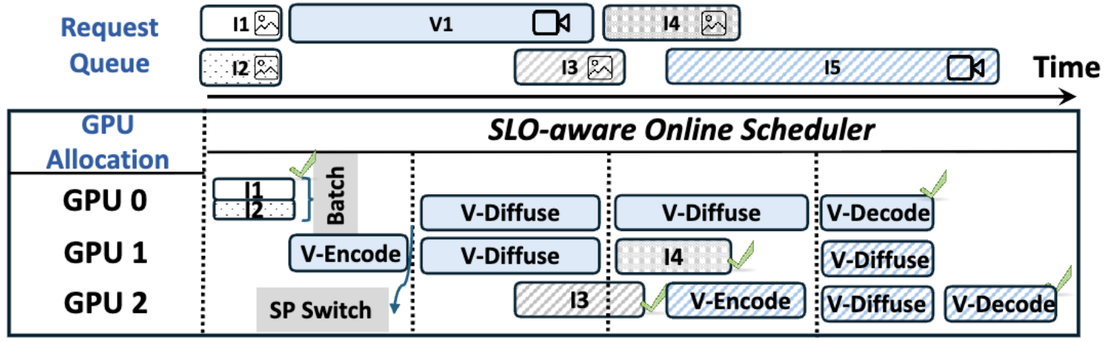


Figure 9: Adaptive resource allocation on the cluster serving mixed image and video requests. The SLO-aware online scheduler dynamically batches same-resolution images, adjusts the video SP degree at boundaries, and decouples stages on separate GPUs to maximize utilization.

Adaptive Resource Allocation (2)

SLO-aware Dynamic Batching (T2I)

Batching Formation: same-resolution of images.

Deadline Check: Profiler will predict e2e latency under enlarged batch size.

Batch Deadline: Depends on earliest deadline in batch.

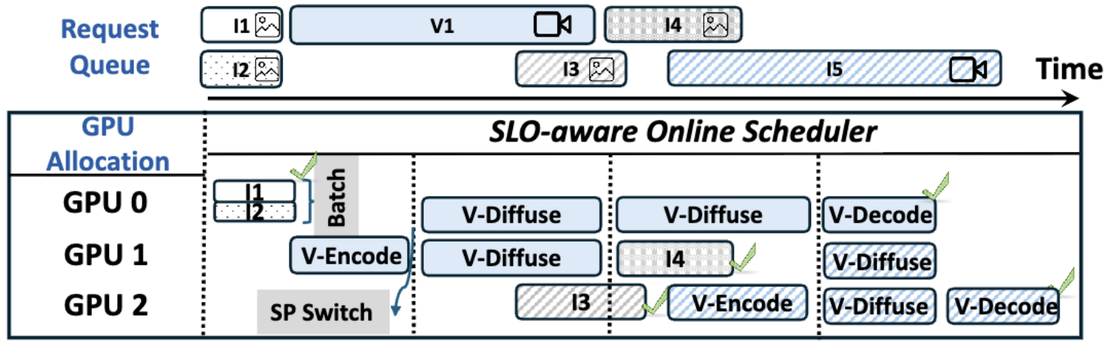


Figure 9: Adaptive resource allocation on the cluster serving mixed image and video requests. The SLO-aware online scheduler dynamically batches same-resolution images, adjusts the video SP degree at boundaries, and decouples stages on separate GPUs to maximize utilization.

SLO-Aware Online Scheduler (1)

Problem Formulation: $\max \sum_{r \in R_I \cup R_V} z_r.$

Constraint: $\sum_r |X_r(t + \Delta_{\text{round}})| \leq N.$

Total GPU allocation cannot exceed cluster size N at any round.

Table 4: Key notation in the §4.4.

Symbol	Description
$\mathcal{G}, N$	GPU set $\{1, \ldots, N\}$ and cluster size
R_I, R_V	Image and video request sets
D_r, A_r	Deadline and arrival time of request r
$S_v^{\text{rem}}(t)$	Remaining steps of video v at time t
$X_r(t)$	GPU alloc. of request r at round boundary t
P	Set of valid SP degrees
Δ_{round}	Scheduling round interval (in denoising steps)
$T_{\text{img}}(b, w, h)$	Profiled latency for image batch of size b
$T_{\text{step}}(w, h, F, p)$	Profiled latency for one video denoising step
$\mathcal{C}_v(t)$	Candidate set for video v at time t
$w(c), f(c)$	GPU cost and score of candidate c
$\ell_v(c, t)$	Projected laxity of video candidate c

SLO-Aware Online Scheduler (2)

Three-Stage DP Algorithm: (No more details)

Stage-1: Build scored candidates.

Stage-2: Knapsack DP over video groups.

Stage-3: Select best plan.

Algorithm 1: SLO-aware DP Scheduler

Input: round time t, GPU set $\mathcal{G}$, image set $I(t)$, video set $\mathcal{V}(t)$, current allocations $\{X_r(t)\}$
Output: Allocation plan $\Pi^\star$

$t^+ \leftarrow t + \Delta_{\text{round}}$;
/* Stage 1: Build scored candidates */
for $g = 0$ **to** N **do**
  $\mathcal{C}_g^{\text{img}}(t) \leftarrow \textsc{EdfBatch}(I(t), g)$ // Eq. (6)
foreach $v \in \mathcal{V}(t)$ **do**
  $\mathcal{C}_v(t) \leftarrow \textsc{GenVideoCandidates}(v, t, X_v(t), \mathcal{G})$ // Eq. (7)
/* Stage 2: Knapsack DP over video groups */
Initialize dp[0][0] $\leftarrow (0,0)$;
for $j = 1$ **to** $|\mathcal{V}(t)|$ **do**
  for $b = 0$ **to** N **do**
    dp$[j][b] \leftarrow (-\infty, -\infty)$;
    foreach $c \in \mathcal{C}_{v_j}(t)$ **do**
      if $w(c) \le b$ ***and*** *no GPU overlap with prev. selections* **then**
        $s \leftarrow$ dp$[j-1][b-w(c)] +$ (recoverable(c), $f(c)$);
        if $s >$ dp$[j][b]$ **then** update dp$[j][b]$, record backpointer ;
/* Stage 3: Select best plan with img. eval. */
foreach *terminal state* b **do**
  $g_{\text{free}} \leftarrow N - b$; combine video score with $\mathcal{C}_{g_{\text{free}}}^{\text{img}}(t)$;
$(b^\star, \mathcal{S}^\star) \leftarrow \textsc{BacktrackBestSolution}(\text{dp})$;
$\Pi^\star \leftarrow \textsc{MaterializeAllocations}(\mathcal{S}^\star, \{X_r(t)\}, \mathcal{G})$;
return $\Pi^\star$;

SLO-Aware Online Scheduler (3)

Time Complexity: $O(G \cdot N \cdot \bar{K})$,
G: video groups, K: average candidates per group.

Table 6: DP solver wall-clock decision time versus number of concurrent request groups G (N=8 GPUs, $\bar{K} \leq 4$ candidates per group). Overhead is relative to a 720p base step (781 ms). Statistics are aggregated across all E1 runs in §6.2.

Concurrent Groups (G)	Mean (ms)	Max (ms)	Overhead/Step (%)
1–2	0.30	0.7	0.09
3–4	0.24	1.0	0.13
5–6	0.25	1.5	0.19
7–8	0.31	1.8	0.23
9–12	0.41	1.9	0.24

03

Evaluation

End-to-End Performance on 8x NVIDIA RTX PRO 6000 GPUs (Single Node)

End-to-End SLO Attainment (1)

1) Different SLO Scale. (Fig. 10)

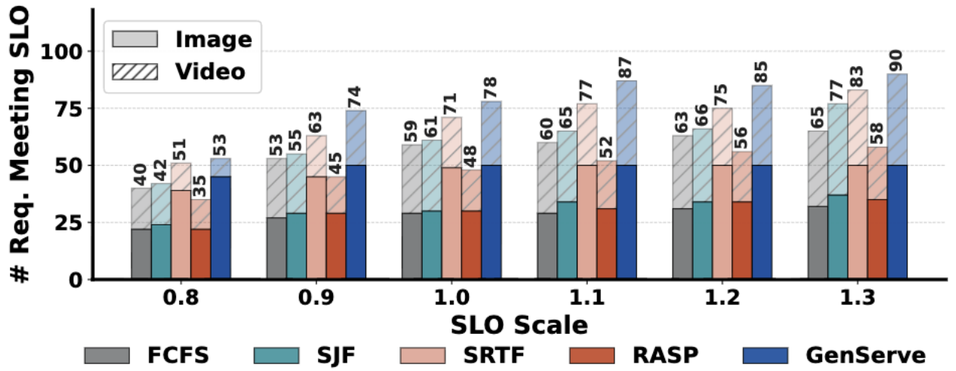


Figure 10: Number of requests meeting SLO versus SLO scale σ at the anchor configuration (balanced mix, 24 req/min). Bars are split into image (solid) and video (hatched) contributions. GENSERVE leads at every σ, especially reaching 90 out of 100 at σ=1.3.

2) Different Workload Mix. (Fig. 11)

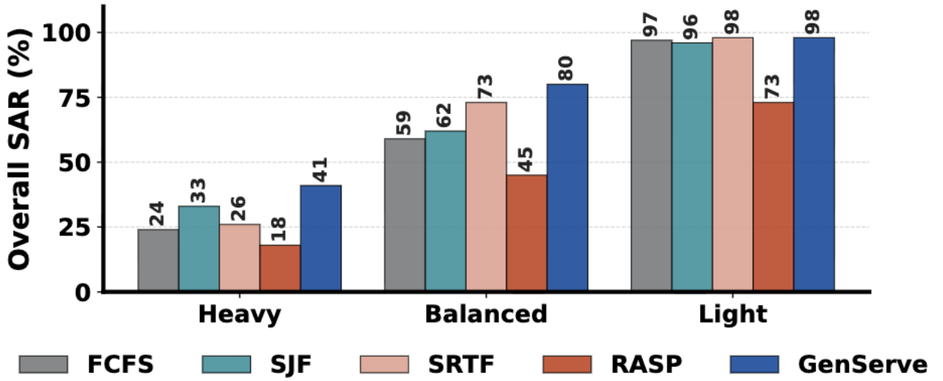


Figure 11: Overall SAR (%) across workload mixes. GENSERVE's advantage grows as the video proportion increases.

End-to-End SLO Attainment (2)

3) Different Arrival Rates. (Fig. 12)

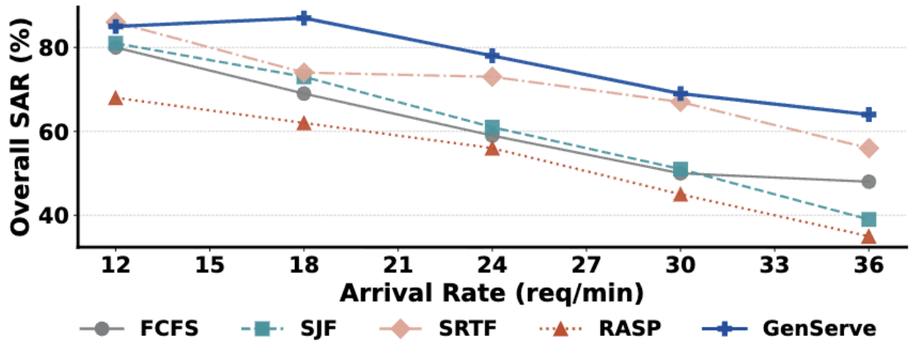


Figure 12: Overall SAR (%) versus arrival rate (12–36 req/min) at σ=1.0 and balanced mix. GENSERVE maintains the highest SAR across all load levels.

4) End-to-End Latency. (Fig. 13)

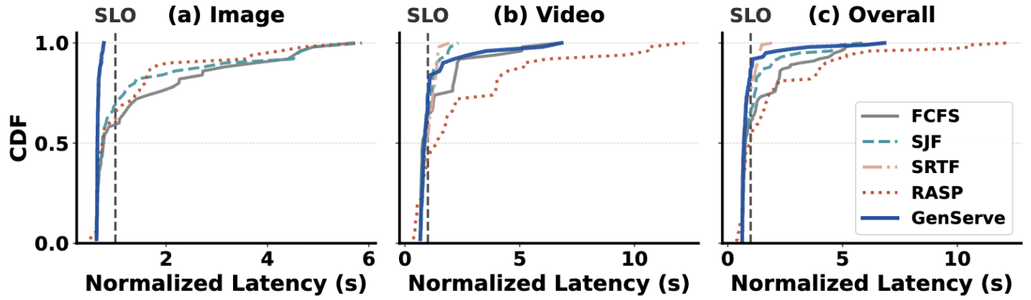


Figure 13: CDF of per-request turnaround latency at the default configuration (σ=1.0, balanced mix, 24 req/min).

Ablation Study

Configuration	Overall SAR	Image SAR	Video SAR	Avg. Image Wait
Baseline (FCFS)	20%	20%	20%	15.9 s
+ Preemption	39%	68%	10%	4.6 s
+ DP Solver	58%	94%	22%	0.2 s
+ SP Switching (GenServe)	63%	94%	32%	0.1 s

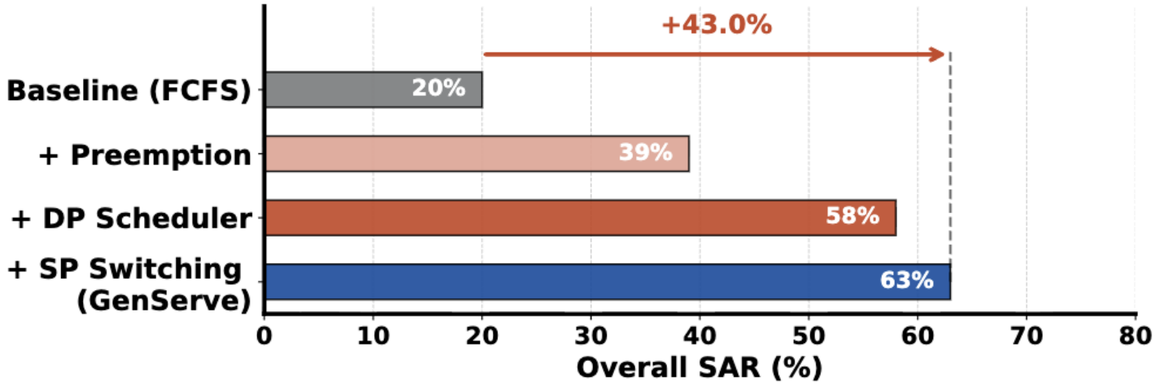


Figure 14: Ablation study under the skewed resolution setting (σ=1.0, balanced mix, 24 req/min). Each bar group adds one component cumulatively.

Sensitivity Analysis (1)

1) Preemption Overhead (Table 7)

Table 7: Preemption overhead summarized by 720p video with different sequence parallelism (SP) degrees. Results are averaged across 5 independent runs.

Res.	SP	Base Step (ms)	Pause (μs)	Resume (ms)	Resume/Step (%)
720p	1	781.57	3.40	0.036	0.005
	2	780.16	3.05	0.464	0.060
	4	780.66	3.00	0.592	0.076
	8	781.15	3.05	0.868	0.112

2) DP Scheduler Overhead (Table 6)

Table 6: DP solver wall-clock decision time versus number of concurrent request groups G (N=8 GPUs, $\bar{K} \leq 4$ candidates per group). Overhead is relative to a 720p base step (781 ms). Statistics are aggregated across all E1 runs in §6.2.

Concurrent Groups (G)	Mean (ms)	Max (ms)	Overhead/Step (%)
1–2	0.30	0.7	0.09
3–4	0.24	1.0	0.13
5–6	0.25	1.5	0.19
7–8	0.31	1.8	0.23
9–12	0.41	1.9	0.24

Sensitivity Analysis (2)

3) Paused Video Memory Overhead (Table 8)

Table 8: GPU memory footprint of a single paused `VideoState` (81 frames, SP=1). Latent and mask tensors are float32; prompt embeddings are bfloat16.

Resolution	Latent (MB)	Mask (MB)	Embeds (MB)	Total (MB)
256p	1.3	1.3	3.5	6.2
480p	4.6	4.6	3.5	12.8
720p	11.8	11.8	3.5	27.2

04

Conclusion & Plan

Conclusion

In this work, we presented **GENSERVE**, a heterogeneity-aware co-serving system that leverages the inherent predictability and step-level preemptibility of diffusion models to maximize SLO attainment for mixed image and video workloads.

GENSERVE combines **intelligent preemption**, **elastic sequence parallelism**, and **dynamic batching** with an **SLO-aware DP-based scheduler** that jointly coordinates resource allocation across all concurrent requests.

Single-Node to Distributed Production Serving (1)

Step 1: Multi-Node Scaling & State Migration

1) SP across nodes, replace NCCL-only USP with low-latency cross-node transport -> Dynamo NIXL ?
2) Migratable preempted *VideoState* across nodes. (Paused video on Node A can resume on Node B)
3) Topology-aware SP grouping that respects NVLink islands. -> Dynamo Grove on Kubernetes?

Step 2: Stage Disaggregation of DiT Pipeline

Decouple Text Encoder / DiT denoising / VAE Decode into independently scaled GPU pools.
Similar to PD disaggregation of LLM.

Step 3: Hierarchical State & Cache Reuse

1) Tiered VideoState / Diffusion Block Manager (GPU <-> host <-> NVMe) -> Dynamo KV Block Manager.
Then it can lifts the current "<= 1 paused video per GPU" limit by orders of magnitude.
2) Cluster-wide router that exploits resolution affinity + prompt-embedding / step-output **cache locality** -> Dynamo KV-Aware Router.

Single-Node to Distributed Production Serving (2)

Step 4: Closed-Loop Cluster Autoscaling

1) Per-step DP scheduler handles fine-grained allocation -> Dynamo scheduling loop

2) Planner uses diffusion per-step predictability for accurate forecasts -> Dynamo SLO Planner.

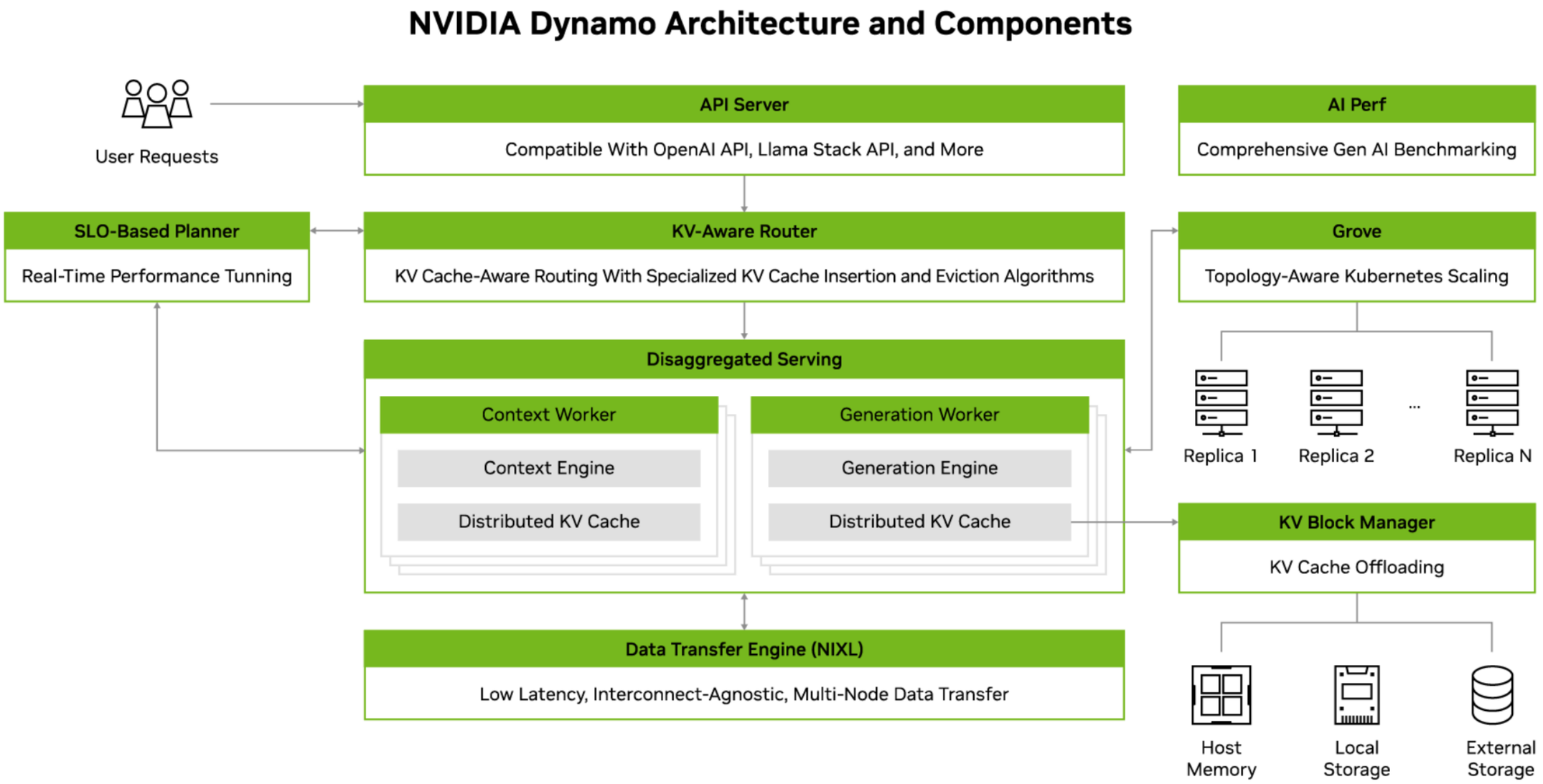
Step 5: Heterogeneous Co-Serving on a Unified Cluster

1) LLM + diffusion on the same cluster — diffusion's predictability fills LLM serving "valleys"

2) Hardware-aware SP-degree selection across mixed pools (H100 / B200 / GB300)

GenServe validates the **algorithmic insights** on a single node; the natural next step is to land them in a production-grade distributed serving framework.

Dynamo Framework



https://developer.nvidia.com/dynamo

Thank You

QUESTIONS & DISCUSSION

Email: fy27@rice.edu