RONDO: SEGMENT-LEVEL CONTINUOUS BATCHING FOR MULTI-TENANT FLOW-VLA SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow-based vision-language-action (VLA) models generate actions via iterative denoising. In multi-tenant serving, diverse robot tasks ranging from reactive obstacle avoidance to precision manipulation naturally require varying numbers of denoising steps. However, executing complete trajectories as indivisible units restricts batching efficiency across differing step counts, while per-step continuous batching incurs prohibitive kernel launch and scheduling overheads. We propose RONDO, a serving framework that partitions denoising trajectories into schedulable segments through a resumable execution abstraction. It combines completion-boundary segmentation with dynamic remaining-work bucketing to prevent segment fragmentation and preserve multi-step execution efficiency. Evaluated with $\pi_{0.5}$ on 8 Ascend 910C NPUs under theoretical load capacity stress testing, RONDO reduces mean serving latency by up to 75.2% relative to Native. In strict real-time closed-loop robot fleet manipulation across 16–32 concurrent robots, RONDO achieves up to 92.3% task success and delivers a 4–$5\times$ improvement over existing serving baselines, sustaining robust physical execution at scale where baseline systems suffer catastrophic buffer starvation.

1 INTRODUCTION

Vision-language-action (VLA) models unify multimodal perception, semantic reasoning, and motor control into an end-to-end policy, mapping raw sensory streams into executable action sequences (Black et al., 2026; Physical Intelligence et al., 2025). In large-scale robot fleet deployments, outfitting each embodied agent with a dedicated high-end accelerator is severely constrained by hardware cost, payload capacity, and thermal dissipation (Kehoe et al., 2015; Shukor et al., 2025). Consequently, real-world robotic systems increasingly centralize policy inference onto shared accelerator clusters—a paradigm known as *multi-tenant serving* (Bansal et al., 2026; Dai et al., 2026). In this shared execution model, unsynchronized request streams from concurrent robots multiplex computing resources without physical hardware partitioning. Server-side queuing delays and serving throughput thereby emerge as the critical bottleneck governing physical control responsiveness, observation freshness, and end-to-end task completion (Gujarati et al., 2020; Black et al., 2025).

Recent state-of-the-art VLA models increasingly adopt flow matching or continuous-time diffusion formulations (Lipman et al., 2023; Chi et al., 2025), generating action chunks through numerical integration over N discrete denoising steps. The step count N directly governs single-request compute cost and serving latency. In multi-tenant robot fleets, heterogeneous step demands arise naturally: reactive maneuvers favor compact denoising horizons for low-latency response, whereas dexterous manipulations require extended horizons for precision. Furthermore, upstream adaptive policies like ProbeFlow (Fang et al., 2026) dynamically assign per-request step counts based on scene complexity. Consequently, the serving system must act as an agnostic substrate, faithfully coordinating concurrent, time-varying step distributions on shared accelerators without sacrificing execution fidelity.

Existing serving systems handle iterative generation through two contrasting paradigms, neither of which accommodates multi-tenant Flow-VLAs. On one hand, conventional serving backends like SGLang (Zheng et al., 2024) employ *monolithic batching*. This paradigm imposes a rigid *same-N constraint*, forcing requests with differing step counts into isolated queues. Furthermore, executing entire trajectories in an indivisible loop locks accelerator cores across all denoising steps, preclud-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ing mid-trajectory admissions and stranding shorter requests behind longer ones. On the other hand, adapting step-level continuous batching from diffusion serving (Xin, 2025; Jiang et al., 2026) allows single-step preemption, but incurs severe host overhead. Frequent kernel launches, KV cache indexing, and tensor regrouping rapidly erode computational gains. More importantly, intermediate denoising latents cannot be executed by physical robots. Realized serving progress occurs only when a request finishes, delivering a usable action chunk and freeing execution slots. Sub-step interruptions that do not alter batch composition therefore introduce redundant scheduling latency without advancing physical control. To serve Flow-VLAs efficiently, the system must decouple generation states while aligning scheduling horizons with meaningful completion boundaries.

To address these challenges, we propose **RONDO**, structuring Flow-VLA serving into a coherent problem-solution pipeline at multi-step segment granularity for multi-tenant embodied robot fleets. First, to enable intra-trajectory regrouping, RONDO introduces a *resumable flow execution* abstraction that decouples generation into persistent accelerator-resident states, pausing and resuming trajectories across batches without host transfers or numerical drift. Second, because completion boundaries guarantee tangible service progress, RONDO aligns batch execution horizons with the *earliest request completion boundary* ($L(B) = \min_{i \in B} r_i$). This guarantees deterministic capacity release, trading a bounded admission wait for mid-segment arrivals for amortized host dispatch. Third, to prevent wide step variance from collapsing segment horizons to single steps (*1-step degradation*), RONDO dynamically partitions requests into remaining-work buckets under high load, preserving multi-step batch amortization. We evaluate RONDO with $\pi_{0.5}$ on 8 Huawei Ascend 910C NPUs across two complementary regimes. Under theoretical load capacity stress testing with synthetic arrivals, RONDO cuts serving latency by up to 75.2% relative to Native. In real-world multi-robot fleet manipulation under strict real-time control constraints (16–32 concurrent robots on LIBERO), RONDO achieves up to 92.3% task success and delivers a 4–$5\times$ improvement over existing serving backends, sustaining reliable physical execution at scale where baseline systems collapse due to buffer starvation.

This paper makes the following contributions:

1. **Resumable flow execution abstraction.** We dismantle the monolithic whole-trajectory barrier in VLA serving by decoupling generation into persistent accelerator-resident states, enabling preemption and resume across dynamic batches with zero host round-trips and strict numerical fidelity.
2. **Segment-level continuous batching framework.** We present RONDO, a scheduling framework governed by request completion boundaries for deterministic capacity release. Coupled with dynamic remaining-work bucketing, RONDO eliminates 1-step degradation and sustains high hardware utilization under varying traffic.
3. **Two-level empirical evaluation with physical task gains.** We establish a two-level evaluation framework spanning server-side theoretical load capacity stress testing and client-side closed-loop fleet manipulation under strict real-time deadlines. Across both regimes on 8 Ascend 910C NPUs, RONDO cuts queuing latency substantially and delivers up to a $5.0\times$ improvement in physical task success over conventional serving systems.

2 BACKGROUND AND PROBLEM ANALYSIS

2.1 FROM SINGLE-REQUEST INFERENCE TO MULTI-ROBOT SHARED SERVING

Flow-based vision-language-action (VLA) models generate action chunks through iterative numerical integration across N denoising steps, where N directly governs single-request computation (Black et al., 2026; Physical Intelligence et al., 2025; Lipman et al., 2023; Chi et al., 2025). Existing efficiency optimizations focus primarily on isolated requests (Hu et al., 2024). Algorithmic approaches like ProbeFlow (Fang et al., 2026) predict adaptive step counts per request, while system frameworks like DistriFusion (Li et al., 2024) and PipeFusion (Fang et al., 2025) parallelize diffusion steps across devices. While these techniques reduce floating-point operations or single-request latency, they do not address the multi-tenant challenge: coordinating concurrent request streams whose step requirements differ and vary over time.

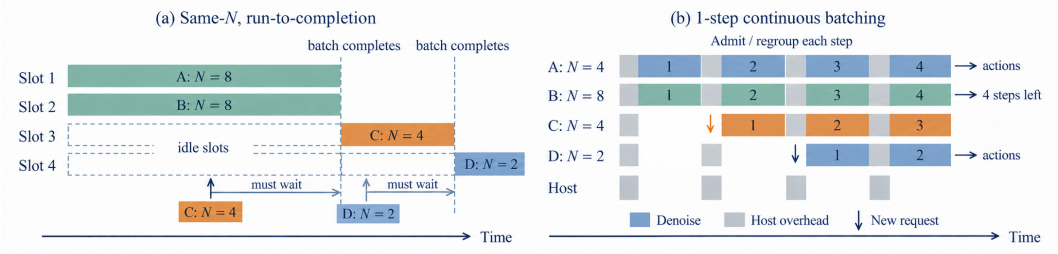

Figure 1: Scheduling extremes: (a) monolithic same-N batches leave slots idle and delay new requests; (b) step-level batching admits requests at each boundary but pays host overhead every step. Timelines are schematic.

2.2 Robot-Level Serving and Action Scheduling

Systems such as Kairos (Dai et al., 2026) and Armory (Bansal et al., 2026) schedule requests based on robot-side execution states and action buffer depletion to avoid control starvation. However, their scheduling flexibility is bounded by the backend engine's execution granularity. If the serving backend executes complete trajectories as indivisible, monolithic units (Zheng et al., 2024), external schedulers cannot preempt or regroup batches during generation. Realizing dynamic scheduling intentions requires an inference engine that exposes intra-trajectory preemption and regrouping boundaries.

2.3 Continuous Batching in Iterative Denoising

Continuous batching has evolved from token-level scheduling in autoregressive language models (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024) to step-level scheduling in diffusion transformers (Xin, 2025; Jiang et al., 2026; Lu et al., 2026; Ye et al., 2026), establishing that generation trajectories need not execute as unbroken units. Unlike image generation, Flow-VLAs produce intermediate latents that cannot be executed by physical robots until all N steps conclude. While per-step continuous batching offers frequent preemption points, triggering rescheduling when no requests arrive or finish introduces redundant host overhead. RONDO structures execution into multi-step segments defined by completion boundaries, striking a principled balance between regrouping flexibility and compute efficiency.

2.4 Problem Analysis: Monolithic vs. Step-Level Batching

Figure 1 contrasts two scheduling extremes in iterative denoising. In monolithic same-N batching (Figure 1a), existing VLA serving backends like SGLang (Zheng et al., 2024) enforce two structural constraints: a *same-N barrier* that segregates requests with differing step counts into isolated queues, and a *monolithic run-to-completion loop* that locks accelerator resources across all N steps. For instance, when a batch occupies only two of four available slots, a shorter request ($N = 4$) arriving midway cannot utilize the vacant capacity and must wait until the entire monolithic batch finishes execution (dashed line), causing acute head-of-line blocking under bursty multi-robot traffic.

Conversely, while step-level continuous batching (Xin, 2025; Jiang et al., 2026) enables admission at every step boundary, Flow-VLAs generate intermediate latents that cannot be streamed to robot actuators until all N steps finish. Figure 1b shows C and D joining successive rounds, with gray blocks marking host scheduling overhead (KV collation, kernel re-launches) at every step, even without new arrivals or completions. This frequent interruption can induce premature queuing collapse under high load (timelines are schematic). This motivates RONDO: eliminating redundant intermediate scheduling while preserving preemption when requests complete and release resources.

3 RONDO Design

RONDO resolves the tension between whole-trajectory blocking and single-step overhead by operating at the granularity of multi-step segments. By decoupling generation state from monolithic

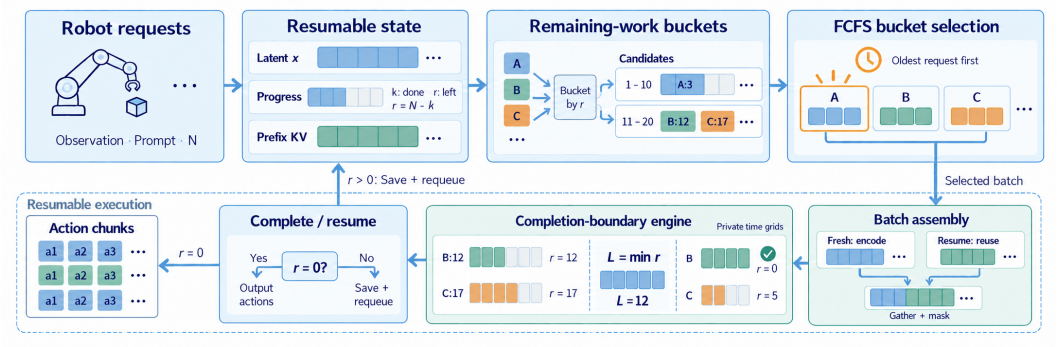

Figure 2: Overview of the RONDO serving architecture and execution lifecycle.

batches into a resumable execution abstraction, RONDO enables intra-trajectory dynamic regrouping while preserving three strict execution invariants: total assigned step counts N_i, discrete time integration grids, and precomputed visual-linguistic conditioning representations. For request i, N_i is its assigned number of denoising steps, k_i is the number already completed, and $r_i = N_i - k_i$ is the number remaining. These count model integration steps, not robot actions. A fresh request starts at $k_i = 0$, $r_i = N_i$; a completed request has $k_i = N_i$, $r_i = 0$. As illustrated in Figure 2, the system is organized into a two-tier pipeline: in the upper row (scheduling tier), persistent request states feed remaining-work candidate organization and FCFS batch selection; in the lower row (execution pipeline), the selected batch is assembled with fresh or cached prefixes, executed to its earliest completion boundary, and routed to client delivery or continuation requeuing (request tiles are schematic).

3.1 RESUMABLE FLOW EXECUTION: DECOUPLED PER-REQUEST STATE

In flow-based VLA policies, synthesizing an action chunk from an initial Gaussian noise sample $x_{i,0} \sim \mathcal{N}(0, I)$ corresponds to numerically integrating a velocity vector field v_θ across N_i discrete steps. For request i, let k denote its local completed-step count (the current value of k_i). Its next update follows Equation 1:

$$x_{i,k+1} = x_{i,k} + \Delta t_{i,k} \cdot v_\theta(x_{i,k}, t_{i,k}, c_i), \tag{1}$$

where $x_{i,k}$ denotes the latent action state, $t_{i,k}$ and $\Delta t_{i,k}$ represent the discrete integration timestep and step size specific to request i, and c_i denotes the conditioning representation derived from visual observations and language instructions.

Instead of binding integration progress to batch-level lifetimes, RONDO decouples state management into an explicit per-request state descriptor S_i (Equation 2):

$$S_i = \big(x_{i,k_i},\ \mathcal{T}_i,\ c_i,\ N_i,\ k_i,\ r_i\big), \tag{2}$$

where $\mathcal{T}_i$ is the prescribed integration grid; k_i indexes the current latent and grid position, and $r_i = N_i - k_i$ tracks the remaining work. For example, $N_i = 20$ and $k_i = 3$ imply $r_i = 17$. After a segment of length $L = 12$, this request has $k_i = 15$ and $r_i = 5$, and resumes from its own grid position. At each step boundary, intermediate action latents and integration progress advance dynamically, while the conditioning context c_i (vision and language KV embeddings) is read-only and preserved in accelerator memory across batch transitions. When batch regrouping occurs, the runtime updates device tensor views and batch indexing directly in accelerator memory without host-device data transfers or prompt re-encoding.

3.2 DYNAMIC REMAINING-WORK BUCKETING

While completion-boundary segmentation ensures deterministic capacity release, co-batching requests with widely disparate remaining work can compromise execution efficiency. Specifically, if a request on its final step ($r_i = 1$) is co-batched with requests requiring substantially more steps ($r_i \gg 1$), the batch execution horizon prematurely collapses to a single step ($L(B) = \min_{i \in B} r_i =$

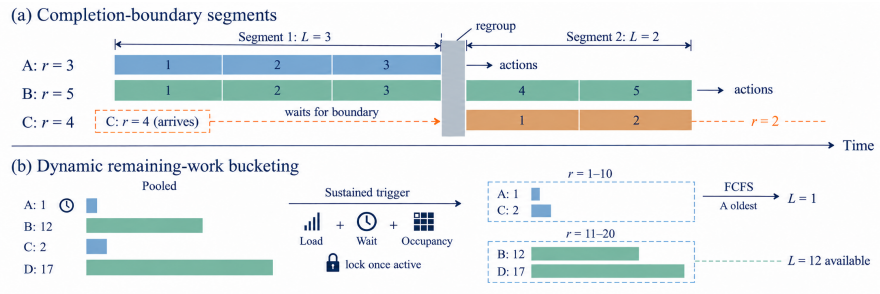

Figure 3: Schematic RONDO mechanisms. (a) A three-step segment completes A; B resumes with two remaining steps and joins waiting C. (b) Sustained load, waiting time, and occupancy trigger locked remaining-work buckets; FCFS selects the candidate with the oldest original arrival (A here).

1), re-introducing frequent host scheduling interrupts and eroding compute amortization. While grouping requests into separate remaining-work queues eliminates this degradation, statically partitioning queues under light or moderate traffic fragments the candidate pool: each partition receives too few requests, resulting in overly small batch sizes and low accelerator compute utilization. To balance batch occupancy and segment length, RONDO introduces *dynamic bucketing*. It operates in a unified pooled mode under low load to aggregate requests and maximize hardware utilization, and dynamically activates remaining-work partitioning only when arrival pressure and queue depth are sufficient to sustain dense batches within individual buckets.

Let $z \in \{0, 1\}$ denote the bucketing state. At each scheduling boundary, fresh requests ($r_i = N_i$) and continuations are assigned an effective bucket identifier

$$\text{bucket}(r_i; z) = \begin{cases} 0, & z = 0, \\ \lfloor (r_i - 1)/w \rfloor, & z = 1, \end{cases} \tag{3}$$

where w is the bucket width. In pooled mode, compatible requests can batch across remaining-work ranges. Once bucketing is active, requests batch within their current range; a continuation with $r = 17$ that advances to $r = 5$ moves from the 11–20 bucket to 1–10 when $w = 10$. The candidate organization in Figure 2 illustrates this bucketed mode.

Workload-Driven Activation and Locking. To detect saturation, the scheduler continuously monitors incoming workload intensity:

$$W_{\text{raw}}(t) = \frac{1}{T_{\text{win}}} \sum_{i \in \mathcal{A}_{\text{fresh}}(t - T_{\text{win}}, t]} N_i, \tag{4}$$

where $\mathcal{A}_{\text{fresh}}$ denotes the set of fresh arrivals within a sliding window T_{win}; continuations are excluded to track net external demand. The raw rate is smoothed via an exponential moving average with time constant τ_{ema} to yield $\overline{W}(t)$. The controller activates bucketing ($z \leftarrow 1$) when three joint conditions persist for duration Δt_{act}: (1) smoothed workload reaches capacity threshold $\overline{W}(t) \geq \theta_{\text{work}}$; (2) queuing delay reaches latency tolerance $\max_{i \in \mathcal{Q}_{\text{ready}}} t_{\text{wait},i} \geq \theta_{\text{wait}}$; and (3) an *occupancy safeguard* confirms that at least M_{min} distinct remaining-work buckets contain at least K_{min} compatible candidate requests. This safeguard prevents premature queue partitioning when arrival volume is high but per-bucket candidate densities are sparse. Once activated, the state is locked for the current serving regime to eliminate thrashing near the operating boundary. Across both pooled and bucketed modes, candidate batches are scheduled in first-come, first-served (FCFS) order based on the arrival timestamp of their oldest pending request (with continuations retaining their original timestamps), guaranteeing fairness and preventing starvation.

Figure 3 illustrates both mechanisms. In (a), A and B have three and five remaining steps, so the selected batch executes a segment of length $L = 3$. A completes and exits; B retains two remaining steps and regroups with waiting C, giving the next segment $L' = 2$. In (b), requests initially share a pooled candidate set. Sustained workload, waiting-time, and occupancy conditions activate bucketing, which remains locked for the serving regime. With $w = 10$, requests with $r = 1, 2$ and $r = 12, 17$ form separate candidates. Assuming A has the oldest original arrival, FCFS selects its candidate first; the longer-work candidate retains a potential twelve-step segment. Unlike autoregressive token serving (Kwon et al., 2023; Zheng et al., 2024), intermediate Flow-VLA latents cannot be returned as executable actions.

4 IMPLEMENTATION

Decoupled Serving Pipeline Stages. We implement RONDO atop SGLang's `Pi05Pipeline` (v0.5.19) on 8 Huawei Ascend 910C NPUs (512 GiB aggregate HBM, TP $= 8$, DP $= 1$) with `torch-npu` and `sgl-kernel-npu`, preserving pre-trained weights and native kernels. Three decoupled stage adapters govern request lifecycles: `VLAPrefixEncodingStage` evaluates ViT visual backbones and LLM conditioning prefixes solely for fresh arrivals ($k = 0$), extracting conditioning representations c_i; continuations ($k > 0$) completely bypass prefix encoding and reuse resident KV caches. `VLAActionDenoisingStage` replaces monolithic sampling loops with a resumable flow engine executing $L(B)$-step segments via in-place Euler forward updates ($x_i \leftarrow x_i + \Delta t_{i,k} \cdot v_\theta$). Finally, `VLAActionPostprocessStage` unbinds finished requests ($r_i = 0$) to dispatch action chunks via HTTP, while requeuing incomplete continuations ($r_i > 0$) back to the scheduling candidate pool.

In-Device State Persistence and Distributed Synchronization. Each in-flight request binds to a `SegmentFlowState` tracking latents $x_{i,k}$, integration grid $\mathcal{T}_i$, step counters (N_i, k_i, r_i), and KV contexts. Intermediate action latents and conditioning KV caches persist directly in device HBM as non-owning tensor views. Recombining continuing requests with newly admitted requests is performed entirely within accelerator memory via `torch.cat` across batch dimensions, avoiding host-device PCIe round-trips. Under tensor-parallel execution (TP $= 8$), Rank 0 serves as the centralized coordinator: it maintains the dynamic bucketing state, selects candidate batches via original-arrival FCFS, and broadcasts compact execution manifests (request IDs, segment length $L(B)$, and padding dimension $T_{\max} = \max_{i \in B} T_i$) to synchronize worker ranks. Fresh requests right-pad prefix lengths to $T_{\max}$ with boolean attention masks. Continuing requests bypass the 50 ms batch accumulation timeout to prevent queue bubbles.

Profiling-Guided Parameter Calibration. Through offline profiling of NPU saturation throughput across step mixtures, we observe a saturation crossover at $\approx$ 180–200 denoising steps/s on 8 Ascend 910C NPUs, below which pooled batching maximizes occupancy and above which bucketing prevents 1-step collapse. We accordingly set the workload threshold $\theta_{\text{work}} = 190$ steps/s ($T_{\text{win}} = 10$ s), queuing delay tolerance $\theta_{\text{wait}} = 0.5$ s, occupancy safeguard $K_{\min} = 8$ ($M_{\min} \geq 2$), and bucket widths $w = 10$ (N2–16) and $w = 16$ (N4–32).

5 EMPIRICAL EVALUATION

5.1 EXPERIMENTAL METHODOLOGY AND SETUP

Hardware, Policy, and Infrastructure. Experiments run on dual HiSilicon Kunpeng 920 CPUs and 8 Huawei Ascend 910C NPUs (512 GiB HBM, TP $= 8$, DP $= 1$) with SGLang v0.5.19 and PyTorch v2.10.0 on openEuler 22.03. We evaluate $\pi_{0.5}$ (Physical Intelligence et al., 2025) fine-tuned for LIBERO (Liu et al., 2023) manipulation (Robosuite (Zhu et al., 2025), MuJoCo (Todorov et al., 2012)), using receding horizon control with 50-step action chunks (10 executed per cycle) with OSMesa CPU rendering.

Workloads and Baselines. We evaluate workloads across two step-budget distributions: a moderate 8-tier spectrum with $N \in \{2, 4, 6, 8, 10, 12, 14, 16\}$ and an extended wide spectrum spanning $N \in [4, 32]$ with up to 12 tiers. Both distributions are evaluated under two distinct regimes: (1) *Theoretical load capacity stress tests* driven by synthetic Poisson arrivals across varying arrival intensities (4–22 req/s for N2–16; 4–14 req/s for N4–32); and (2) *Strict real-time closed-loop fleet simulation* on LIBERO manipulation tasks across 16–32 concurrent robots at 1.0–2.0 Hz control rates. We benchmark against: (1) **Native** (Zheng et al., 2024), standard SGLang with same-N batching and monolithic execution; (2) **Iter-CB** (Xin, 2025), step-level continuous batching ($L = 1$); and (3) **Kairos** (Adapted) (Dai et al., 2026). Specifically, Kairos prioritizes requests based on robot-side action buffer depletion to mitigate starvation, but operates atop a monolithic backend without intra-trajectory preemption.

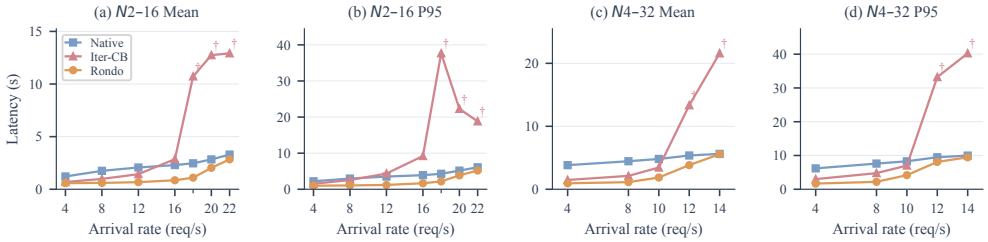

Figure 4: Open-loop mean and P95 response latency for $N2$–16 (a–b) and $N4$–32 (c–d). RONDO uses dynamic bucketing with FCFS. †: nonzero client drops; dropped arrivals are excluded from latency. Each point is one run.

5.2 SERVING LOAD CAPACITY AND THEORETICAL SATURATION LIMITS

We evaluate open-loop serving dynamics and throughput saturation under synthetic Poisson arrival traces across load intensities (60 s warm-up, 300 s measurement, max batch size 32, client in-flight limit 256). Client requests carry representative observation contexts and fixed noise seeds over HTTP to evaluate server-side queuing latency, request throughput, and drop rates under controlled arrival pressure. RONDO operates with dynamic bucketing and original-arrival FCFS (Section 3.2), locking the bucketing state once activated under sustained load.

Mitigating Head-of-Line Blocking at Moderate Load. As shown in Figure 4, RONDO consistently delivers lower mean and P95 latencies than Native across non-saturating regimes. Under light-to-moderate traffic (4–12 req/s), Native suffers from the *same-N barrier*: its monolithic execution requires batches to share identical step budgets N_i, fragmenting traffic across 8 ($N2$–16) or 12 ($N4$–32) isolated queues. Deep-budget batches ($N = 16$ or 32) monopolize NPUs for seconds, forcing newly arrived short requests ($N = 2$ or 4) to wait idly behind them (head-of-line blocking). In contrast, RONDO's resumable execution and dynamic prefix masking enable requests across diverse step counts and context lengths to co-execute in high-occupancy batches. Shorter requests complete early, release batch slots, and return actions immediately. At 4 req/s under $N4$–32, RONDO cuts mean latency from 3.828 s to 0.949 s (a **75.2% reduction**), maintaining flat, sub-second responses while Native stays above 3.8 s.

Host Dispatch Bottleneck vs. Segment Amortization at High Load. As arrival rates approach capacity (16–22 req/s on $N2$–16; 10–14 req/s on $N4$–32), a stark divergence separates RONDO and Iter-CB. Iter-CB continuously batches at single-step granularity ($L = 1$). However, Flow-VLA intermediate latents are continuous vector fields unexecutable by physical robots. Preempting at $L = 1$ forces the host CPU to traverse queues, slice attention masks, and launch kernels after *every single step*. Cumulative host dispatch latency outpaces kernel duration, starving NPUs and triggering exponential queuing. Under $N2$–16 at 18 req/s, Iter-CB's latency explodes to 10.721 s with 3.14% drops (12.919 s / 13.16% drops at 22 req/s); on $N4$–32 at 14 req/s, it collapses to 21.591 s with 17.97% drops. In contrast, RONDO's completion segmentation ($L(B) = \min r_i$) executes multi-step segments until at least one request finishes, amortizing host overhead over compute-dense kernels. Combined with dynamic bucketing ($w = 10$) preventing 1-step degradation, RONDO maintains stable latency (1.100 s at 18 req/s on $N2$–16; 5.575 s at 14 req/s on $N4$–32) with **zero client drops**.

5.3 CLOSED-LOOP MULTI-ROBOT SIMULATION UNDER STRICT REAL-TIME CONTROL

We deploy our serving engine into closed-loop LIBERO simulation under strict real-time control across 16–32 concurrent robots at 1.0–2.0 Hz control rates. Robots operate with asynchronous receding horizon control, executing actions at physical wall-clock step intervals ($\Delta t_{\text{action}} = 1/\text{Hz}$) while asynchronously prefetching the next action chunk in the background. If extended server queuing starves the robot's local action buffer before the next response arrives, the robot stalls and the episode terminates as a timeout failure. Table 1 reports results under narrow $N2$–16 and wide $N4$–32 step budgets (60 s warm-up, 180 s measurement; runs falling below 30% success early-stop).

Moderate Step Spectrum ($N2$–16): Concurrency and Control Cadence. Under $N2$–16 (Table 1, top), Native's per-N queuing causes rigid serving latencies ($\approx$ 2.7–3.3 s) that leave action

Table 1: Strict real-time LIBERO results at 1.0–2.0 Hz for two step-budget distributions. Each cell is task success (%) ↑ / mean request latency (s) ↓.

N	c	Hz	Native	Iter-CB	Kairos	RONDO
2–16	16	1.0	81.8 / 2.876	**92.9** / 0.960	70.0 / 2.688	85.7 / **0.746**
		1.5	13.3 / 2.717	57.9 / 0.990	12.5 / 2.678	**66.7** / **0.790**
		2.0	14.3 / 2.702	29.6 / 0.964	0.0 / 2.742	**66.7** / **0.798**
	24	1.0	50.0 / 2.896	**83.3** / 1.340	50.0 / 2.951	82.6 / **1.002**
		1.5	—	48.4 / 1.281	—	**75.0** / **1.073**
		2.0	—	—	—	**62.5** / **1.073**
	32	1.0	55.6 / 3.275	73.3 / 1.633	27.3 / 3.162	**78.9** / **1.346**
		1.5	—	19.0 / 1.522	—	**53.3** / **1.349**
		2.0	—	—	—	**48.3** / **1.377**
4–32	16	1.0	25.0 / 5.160	50.0 / 1.829	25.0 / 5.119	**92.3** / **1.083**
		1.5	0.0 / 5.039	19.4 / 1.664	3.1 / 4.689	**75.0** / **1.092**
		2.0	1.8 / 3.736	3.6 / 1.597	1.1 / 3.530	**35.0** / **1.129**
	24	1.0	—	32.1 / 2.164	—	**80.0** / **1.494**
		1.5	—	—	—	**56.5** / **1.511**
		2.0	—	—	—	**41.7** / **1.479**
	32	1.0	—	20.0 / 2.552	—	**81.2** / **1.895**
		1.5	—	—	—	**46.2** / **1.888**
		2.0	—	—	—	**22.2** / **1.890**

c: concurrent robots; **bold**: best measured value per metric within each row. —: unmeasured after early stopping, not zero. The success threshold for early stopping is 30%.

buffers vulnerable to starvation under higher control rates (13.3% success at 16 robots / 1.5 Hz; early-stopped at 24/32 robots). Kairos prioritizes requests by remaining buffer time, but remains bottlenecked by monolithic same-N batches (12.5% success at 1.5 Hz). Iter-CB achieves high success at light load (92.9% at 16 robots, 1.0 Hz), but at 1.5 Hz or 32 robots, prefetch bursts generate single-step dispatch storms that inflate latency to 1.522 s, dropping success to 19.0%. In contrast, RONDO delivers fresh chunks with consistent sub-second to low-latency pacing (0.790 s at 16 robots / 1.5 Hz; 1.002 s at 24 robots / 1.0 Hz), reliably replenishing buffers and preserving observation freshness to achieve **66.7% success at 16 robots / 1.5 Hz** ($5.0\times$ over Native) and sustaining **53.3% at 32 robots / 1.5 Hz**.

Wide Step Spectrum ($N4$–32): Stress-Testing Real-World Step Variance. Under the wide $N4$–32 suite (Table 1, bottom), spanning eight discrete step tiers $\{4, 8, \ldots, 32\}$, the architectural flaws of the baselines become severely amplified: (1) *Native and Kairos Collapse under Queue Fragmentation:* Partitioning traffic across 8 tiers fragments queues, dropping batch occupancy and inflating latency beyond 5.0 s. At 1.5 Hz (16 robots), Native drops to **0.0% success** and Kairos to 3.1%, early-stopping across all higher concurrencies. (2) *Iter-CB Dispatch Multiplication:* For deep trajectories ($N = 32$), Iter-CB performs up to 32 host interrupts per request. Multi-robot dispatch storms drive latency to 1.664–2.552 s, dropping success to 19.4% (16 robots, 1.5 Hz) and 20.0% (32 robots, 1.0 Hz). (3) *Rondo Robustness:* Combining completion segmentation with dynamic bucketing ($w = 16$), RONDO co-batches heterogeneous steps while shielding long trajectories from 1-step fragmentation, maintaining predictable serving latencies ($\approx$ 1.08–1.09 s at 16 robots). It delivers **92.3% success at 1.0 Hz** and **75.0% at 1.5 Hz** (nearly $4\times$ over Iter-CB's 19.4%, while Native collapses completely to 0.0%). At 24 and 32 robots, RONDO maintains **80.0% and 81.2% success at 1.0 Hz**, and **56.5% and 46.2% at 1.5 Hz**, where all baselines early-stopped.

5.4 HOST SCHEDULING OVERHEAD

Host Scheduling Overhead and Resident State Persistence. Figure 5a decomposes host overhead at 8 and 20 req/s. State saves (0.18 ms) and grid setup (0.73 ms) remain sub-millisecond, while tensor regrouping (2.03 ms) scales smoothly. Total host overhead accounts for merely 1.4%–4.1% of segment execution time ($\approx$ 152 ms). This efficiency stems from: (1) **resident state persistence**, retaining latents and KV contexts in device HBM as views recombined via `torch.cat` without host round-trips; and (2) **single-master coordination**, where Rank 0 evaluates queues and broadcasts compact manifests to synchronize workers.

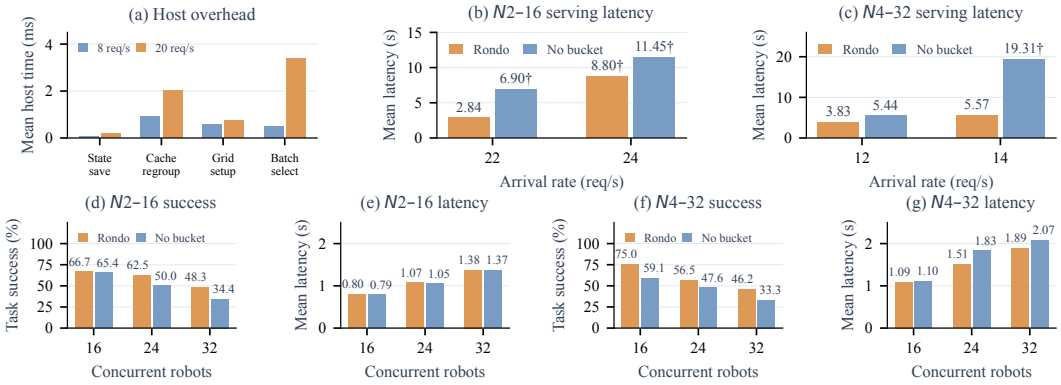

Figure 5: Host overhead (a) and bucketing ablations: serving latency under synthetic arrivals (b–c; †: client drops), and LIBERO success/mean latency with 16/24/32 robots for $N2$–16 at 2.0 Hz (d–e) and $N4$–32 at 1.5 Hz (f–g).

5.5 Component Ablations

Dynamic Bucketing in Theoretical Saturation Regimes. To isolate dynamic bucketing under theoretical load capacity stress testing, Figure 5b–c compares RONDO against an unbucketed baseline (*No bucket*) on matched arrival traces. Without bucketing, indiscriminate pooling mixes finishing requests ($r_i = 1$) with deep continuations ($r_i \gg 1$), collapsing segment horizons to $L(B) = \min r_i = 1$. This *1-step degradation* destroys execution amortization and triggers dispatch storms. On $N4$–32 at 14 req/s (Figure 5c), unbucketed latency reaches 19.31 s with 8.50% drops. Dynamic bucketing ($w = 10$) segregates requests into remaining-work bands to preserve multi-step segments ($L \geq 2$), cutting latency by **71.1%** (to 5.57 s) and eliminating drops. On $N2$–16 at 22 req/s (Figure 5b), bucketing cuts latency by 58.8% (6.90 to 2.84 s); at 24 req/s, it cuts drop rate from 9.44% to 2.03%.

Closed-Loop Control Cadence and Buffer Protection. Under $N2$–16 at 2.0 Hz (Figure 5d–e), RONDO achieves 66.7%, 62.5%, and 48.3% task success at 16, 24, and 32 robots, respectively, versus 65.4%, 50.0%, and 34.4% for *No bucket*. Mean request latencies remain close: 0.798/1.073/1.377 s for RONDO versus 0.785/1.048/1.372 s without bucketing. At 24 and 32 robots, the observed success advantage is 12.5 and 13.9 percentage points, respectively, despite similar mean serving latency. Crucially, this substantial success gap under matched mean latency exposes the peril of mean-latency optimization in real-time robotic serving: unbucketed pooling produces acute horizon jitter, where erratic 1-step flushes interleave with prolonged stalls. While truncated single-step segments yield artificially lower arithmetic latency on paper, their unpredictable delivery intervals cause transient starvation spikes in the robots' local action buffers. In contrast, RONDO's dynamic bucketing enforces regular multi-step chunk replenishment that preserves observation freshness and shields control buffers from starvation. On $N4$–32 at 1.5 Hz (Figure 5f–g), RONDO achieves 75.0%, 56.5%, and 46.2% success at 16, 24, and 32 robots versus 59.1%, 47.6%, and 33.3% for *No bucket*. Corresponding mean latencies are 1.092/1.511/1.888 s versus 1.098/1.828/2.069 s.

6 Conclusion

We presented RONDO, a segment-level continuous batching serving framework tailored for multi-tenant Flow-VLA policies. By unifying resumable flow execution, completion-boundary segmentation, and dynamic remaining-work bucketing, RONDO eliminates both the head-of-line blocking inherent to monolithic same-N batching and the dispatch overheads of single-step continuous batching. Evaluated on 8 Ascend 910C NPUs with $\pi_{0.5}$, RONDO reduces serving latency by up to 75.2% under saturation stress testing and achieves up to 92.3% success in real-time closed-loop fleet manipulation across 16–32 concurrent robots, delivering up to $5.0\times$ higher task completion where baseline serving systems suffer catastrophic buffer starvation. These results demonstrate that aligning batch execution horizons with request completion boundaries provides a scalable, robust systems foundation for large-scale multi-robot embodied intelligence.

AI USE STATEMENT

In this work, the conceptual ideation, technical innovations, and core abstractions—including the resumable flow execution abstraction, completion-boundary segmentation, and dynamic remaining-work bucketing—were conceived and formulated entirely by the authors without the involvement of generative AI. Furthermore, generative AI was not involved in experimental design, baseline parameter configurations, parameter calibrations, or the analysis and interpretation of empirical findings.

Generative AI tools (large language models) were employed strictly as assistive instruments in two practical capacities: (1) **Code development and experimental execution:** assisting with software engineering of the serving framework, writing boilerplate routines for the evaluation harness scripts, and automating experimental execution pipelines; and (2) **Manuscript polishing:** improving grammatical conciseness, phrasing readability across manuscript drafts, and formatting LaTeX table alignments and bibliographic entries. All software implementations, benchmark harnesses, and experimental pipelines were thoroughly reviewed, verified for numerical correctness, and executed on physical Huawei Ascend 910C NPUs by the authors. The authors assume full responsibility for the originality, validity, and integrity of all content presented in this paper.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Rohan Bansal, David He, Nadun Ranawaka Arachchige, Zhenyang Chen, Soobum Kim, Kexin Rong, and Danfei Xu. Action chunk scheduling for batched robot policy serving, 2026. URL https://arxiv.org/abs/2608.00337.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 33383–33407. Curran Associates, Inc., 2025. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/300ccb2187dedd4edcc07f7e76d8e553-Paper-Conference.pdf.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Yinwei Dai, Ganesh Ananthanarayanan, Landon Cox, Xenofon Foukas, Bozidar Radunovic, and Ravi Netravali. Kairos: A scalable serving system for physical ai, 2026. URL https://arxiv.org/abs/2605.11381.

Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. Pipefusion: Patch-level pipeline parallelism for diffusion transformers inference. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 98434–98455. Curran Associates, Inc., 2025. doi: 10.52202/085713-3289. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8da04a60948be713dc766f0c7e3a5b1f-Paper-Conference.pdf.

Zhou Fang, Jiaqi Wang, Yi Zhou, and Qiongfeng Shi. Probeflow: Training-free adaptive flow matching for vision-language-action models, 2026. URL https://arxiv.org/abs/2603.17850.

Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving dnns like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/gujarati.

Xixi Hu, Bo Liu, Xingchao Liu, and Qiang Liu. Adaflow: Imitation learning with variance-adaptive flow-based policies. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 138836–138858. Curran Associates, Inc., 2024. doi: 10.52202/079017-4405. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/fa9cfdb49f7de3dee54007f84c0745b1-Paper-Conference.pdf.

Xiaoxiao Jiang, Suyi Li, Lingyun Yang, Tianyu Feng, Zhipeng Di, Weiyi Lu, Guoxuan Zhu, Xiu Lin, Kan Liu, Yinghao Yu, Tao Lan, Guodong Yang, Lin Qu, Liping Zhang, and Wei Wang. Flashps: Efficient generative image editing with mask-aware caching and scheduling. In *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, pp. 2109–2125, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722127. doi: 10.1145/3767295.3769379. URL https://doi.org/10.1145/3767295.3769379.

Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015. doi: 10.1109/TASE.2014.2376492.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7183–7193. IEEE, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023. URL https://arxiv.org/abs/2210.02747.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

Runyu Lu, Shiqi He, Wenxuan Tan, Shenggui Li, Ruofan Wu, Jeff J. Ma, Ang Chen, and Mosharaf Chowdhury. Tetriserve: Efficiently serving mixed dit workloads. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 1982–1997, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400723599. doi: 10.1145/3779212.3790233. URL https://doi.org/10.1145/3779212.3790233.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_0.5$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033,

2012. doi: 10.1109/IROS.2012.6386109.

Rachel Xin. Dit-serve and deepcoder: Enabling video and code generation at scale. Master's thesis, EECS Department, University of California, Berkeley, May 2025. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-46.html.

Fanjiang Ye, Zhangke Li, Xinrui Zhong, Ethan Ma, Russell Chen, Kaijian Wang, Jingwei Zuo, Desen Sun, Ye Cao, Triston Cao, Myungjin Lee, Arvind Krishnamurthy, and Yuke Wang. Genserve: Efficient co-serving of heterogeneous diffusion model workloads, 2026. URL https://arxiv.org/abs/2604.04335.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning, 2025. URL https://arxiv.org/abs/2009.12293.