arXiv:2611.03103v2 [cs.RO] 16 Nov 2026

VLALAGBENCH: UNDERSTANDING WHEN AND WHY VLA INFERENCE LATENCY MATTERS

Qi Wen [1] **Borui Xu** [1] **Hussein Amro** [1] **Hongshi Tan** [1] **Hanmin Lin** [2] **Chaoyi Ruan** [1] **Shanghao Liu** [1] **Gang Yang** [1] **Yuejiang Liu** [1 3] **Jiafei Duan** [1] **Bingsheng He** [1] **Jason Clemons** [4] **Christos Kozyrakis** [3 4] **Wenqi Jiang** [1]

[1]National University of Singapore [2]Nanyang Technological University [3]Stanford University [4]NVIDIA

ABSTRACT

Vision-language-action (VLA) models offer a promising path toward general-purpose robot manipulation. While inference latency can affect task success in a constantly evolving physical world, the relationship between latency and task success rate remains poorly understood. In this paper, we introduce VLALAGBENCH, a comprehensive benchmark for studying VLA inference latency sensitivity across task types, object speeds, execution modes (synchronous and asynchronous), and VLA policies. Using VLALAGBENCH, we find that even a modest inference latency of 100 ms can nearly halve task success on moving targets, with sensitivity varying substantially across tasks and policies. These differences reflect task-specific requirements for precision and continuous motion. By overlapping inference with action execution, asynchronous execution can recover some of the lost success at the cost of more frequent inference requests. Building on these findings, we develop an empirical model that predicts task success from limited calibration data, achieving root-mean-square prediction errors of only 5.7–7.8% across various VLA policies. Given an inference system's latency and per-robot throughput, these predictions guide the selection of system-specific VLA algorithm configurations to maximize task success. We believe VLALAGBENCH and these findings provide a solid foundation for future robotics algorithm–system co-design.

1 INTRODUCTION

Physical AI has the potential to enable robots to perform diverse tasks autonomously in the real world. Recent advances in vision-language-action (VLA) models have brought this goal closer by predicting robot actions via visual observations and natural-language instructions. These models have demonstrated strong capabilities across a wide range of manipulation tasks, highlighting their potential for general-purpose robotics (Black et al., 2025a; NVIDIA et al., 2025; Fang et al., 2026).

Although VLA models need to respond in real time to keep up with changes in the physical world, the inference latency required for successful task execution remains poorly understood. From a systems perspective, VLA-Perf (Jiang et al., 2026) uses 10 Hz and 100 Hz as reference points for acceptable and high-performance inference, respectively, without evaluating how these rates translate into task success. Similarly, Realtime-VLA (Ma et al., 2025; Yang et al., 2026a; Niu et al., 2026) improves inference and execution speed, but demonstrates the benefits on a limited set of tasks, such as catching a falling pen. From a benchmarking perspective, popular robotic manipulation benchmarks such as LIBERO (Liu et al., 2023) and RoboCasa (Nasiriany et al., 2024) primarily evaluate model generalization across tasks and scenes, but their static environments provide limited insight into how inference latency affects task success in evolving physical environments. While recent benchmarks such as DynamicVLA's DOM, MoveBench, and ReflexBench (Xie et al., 2026; Zhang et al., 2026; Chen et al., 2026b) support manipulation tasks with moving objects, these benchmarks do not systematically evaluate how inference latency affects task success: their evaluations either omit latency, use latencies tied to specific hardware, or cover only a limited set of policies and tasks.

In this paper, **we present VLALAGBENCH, a benchmark for systematically studying how VLA inference latency affects task success**, which provides the foundation for future robotic system design. As shown in Figure 1, VLALAGBENCH examines five factors related to latency sensitivity. (1) *Task type*: grasping a moving object, releasing an object into a moving receptacle, placing an object on a moving surface, and manipulating a moving articulated object. (2) *Object speed*: different target speeds within each task. (3) *VLA policy*: popular models including $\pi_{0.5}$, GR00T N1.7, and MolmoAct2 (Black et al., 2025a; NVIDIA, 2026a; Fang et al., 2026). (4) *Execution mode*:

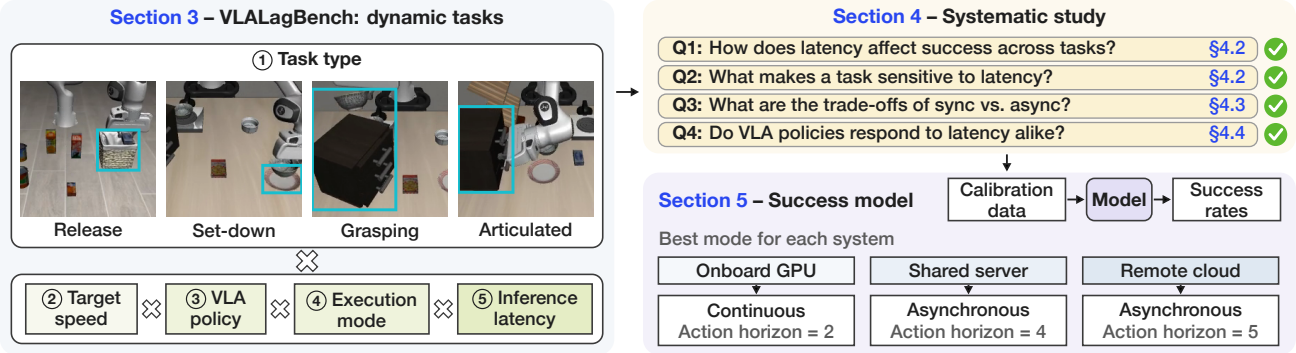


Figure 1. VLALAGBENCH links VLA inference latency to task success through observation staleness and idle time, enabling a compact model to predict success at unseen latencies and choose the execution mode for a given deployment. The benchmark varies five factors—task type, target speed, inference latency, execution mode, and policy—to answer Q1–Q4.

synchronous and various asynchronous execution methods. (5) *Inference latency*: controlled latency levels. Beyond our primary benchmark built on the MuJoCo simulator (Todorov et al., 2012), we also conduct experiments in Isaac Sim (another widely-used simulator) and on real robots to verify our findings.

Our evaluation with VLALAGBENCH yields four **key findings**, each addressing a research question:

Q1. How does inference latency affect task success across different manipulation tasks?
Latency sensitivity varies substantially across tasks (§4.2). For objects moving at 0.1 m/s, a moderate inference latency of 100 ms nearly halves success in one task (from 71.5% to 40.5%), while another retains more (86%) of its zero-latency success rate (from 86.5% to 74.0%).

Q2. What factors determine a task's sensitivity to inference latency?
Latency reduces task success through (1) stale observations (the target has moved by the time inference completes) and (2) robot pauses during synchronous inference. Their impact varies by task: stale observations affect tasks that particularly require precise manipulation, while pauses disrupt tasks requiring continuous motion (§4.2). These factors motivate our next study of asynchronous execution.

Q3. What are the trade-offs between synchronous and asynchronous execution as inference latency increases?
Asynchronous execution overlaps inference with robot actions, keeping the robot moving and benefiting tasks that require continuous motion. When inference latency is sufficiently short, continuous asynchronous execution also reduces observation staleness through more frequent action updates, improving precision. These benefits come at the cost of more frequent inference requests (§4.3).

Q4. Do different VLA models exhibit the same latency sensitivity?
VLA policies differ in their sensitivity to stale observations and robot pauses, thus benefiting to different extents from asynchronous execution. Additionally, the best-performing policy at a given inference latency depends on both the task and target speed (§4.4).

Building on these findings, **we further develop an empirical model that guides the selection of algorithm configurations for a given inference system** (§5). Different combinations of GPUs, networks, and software runtimes result in different inference latencies and per-robot throughput. Consequently, the VLA configuration, including the execution mode and action execution horizon, that maximizes task success may differ across systems. Because exhaustively evaluating this configuration space is prohibitively expensive, we develop a model that estimates task success across latencies, execution modes, and execution horizons, using only a small set of measurements. Given a system's latency and throughput constraints, we thus use these predictions to select the feasible algorithm configuration with the highest predicted task success.

This paper makes three main **contributions**, which together provide a solid foundation for future robotics algorithm–system co-designs:

- We introduce VLALAGBENCH to comprehensively benchmark how VLA latency affects task success.
- We present findings on latency sensitivity across tasks, target speeds, execution modes, and VLA policies.
- We develop an empirical model that predicts task success rate and use it to guide algorithm configuration choice given a user-provided system specification.

2 BACKGROUND AND MOTIVATION

In this section, we introduce VLA models and review existing benchmarks for evaluating their manipulation performance. Existing benchmarks offer limited insight into how inference latency affects task success, motivating our systematic investigation on this important topic.

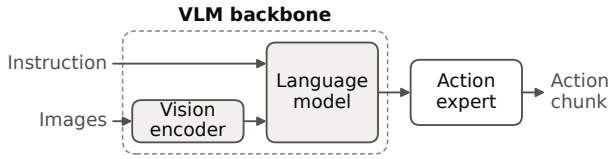


Figure 2. A VLA model combines a VLM backbone with an action expert that predicts a sequence of robot actions.

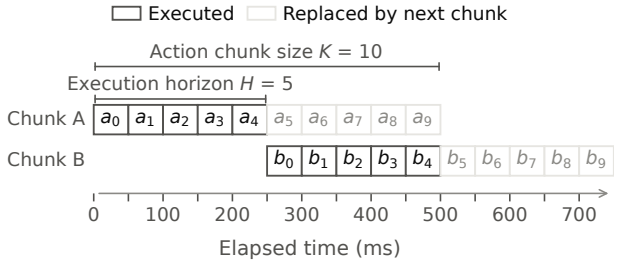


Figure 3. Each VLA inference generates a chunk of actions.

2.1 Vision-Language-Action Models

Vision-language-action (VLA) models represent an important step toward physical AI by using visual observations and natural-language instructions to generate robot actions for continuous control (Black et al., 2025b;a; NVIDIA et al., 2025). As shown in Figure 2, a VLA model combines a vision-language model (VLM) backbone that encodes camera images and language instructions with an action expert that predicts the robot's next sequence of actions. Models such as π_0, $\pi_{0.5}$, GR00T N1, and MolmoAct2 use diffusion transformers (DiT) as action experts, transforming sampled noise into continuous action sequences through multiple refinement iterations (Black et al., 2025b;a; NVIDIA et al., 2025; Fang et al., 2026). OpenVLA and the original MolmoAct instead generate discrete action tokens autoregressively, which are then decoded into robot actions (Kim et al., 2025; Lee et al., 2026).

Modern VLA policies commonly predict an *action chunk*, a sequence of future actions generated in one inference call (Zhao et al., 2023; Black et al., 2025b; NVIDIA et al., 2025; Fang et al., 2026). We denote the number of predicted actions by K, the *action chunk size*, and the number executed before switching to an updated chunk by H, the *execution horizon* (Black et al., 2025c). Predicting multiple actions together helps maintain consistent motion, while executing only a shorter prefix ($H < K$) lets the robot incorporate new observations and adjust its actions sooner (Chi et al., 2023; Black et al., 2025c). Figure 3 shows an example of two chunks generated by separate inference calls (assuming zero inference latency): the robot executes the first $H = 5$ actions of chunk A ($K = 10$), then switches to chunk B, replacing A's remaining actions.

2.2 Robot Manipulation Benchmarks

Widely used robotic manipulation benchmarks evaluate how well policies perform diverse tasks and generalize across objects, goals, and scenes. For example, LIBERO provides task suites that vary spatial relationships, object identities, and task goals, together with a long-horizon suite that requires executing sequences of manipulation skills (Liu et al., 2023). RoboCasa expands evaluation to diverse kitchen environments, covering individual skills such as pick-and-place and opening drawers, as well as composite activities that combine multiple skills (Nasiriany et al., 2024). RoboLab evaluates generalist policies on household manipulation tasks organized around visual recognition, spatial relationships, and procedural skills, and measures their robustness to changes in scene appearance and camera configuration (Yang et al., 2026b).

These benchmarks provide valuable measures of task competence and generalization, but limited insight into the consequences of inference latency. Due to the static physical environments in these benchmarks, a longer inference latency increases task completion time without directly affecting the task outcome (Chen et al., 2026b). Consequently, these benchmarks do not reflect whether a policy can react quickly enough to succeed when the physical world continues to evolve during inference.

2.3 Dynamic Manipulation Benchmarks

Recent dynamic manipulation benchmarks introduce moving objects to test policy reactivity, but they do not systematically isolate how inference latency degrades task success. For example, MoveBench tests grasping under constant, accelerating, and irregular object velocities (Zhang et al., 2026), DynamicVLA's Dynamic Object Manipulation (DOM) benchmark evaluates grasping and placing with moving targets (Xie et al., 2026), and ReflexBench introduces time-critical tasks such as conveyor picking and ball catching (Chen et al., 2026b). However, MoveBench does not account for inference latency during simulation (Zhang et al., 2026). DOM runs the simulator during inference, but its latencies are tied to the policy and GPU used for evaluation, without a controlled sweep showing how task success changes with inference latency (Xie et al., 2026). ReflexBench explicitly varies inference latency under synchronous and asynchronous execution, but studies its effect using a single policy (SmolVLA) on only two tasks (Chen et al., 2026b).

Research gap. Because neither existing benchmarks nor latency-aware model papers (Appendix H) jointly vary task types, object speeds, execution modes, and generalist VLA policies under controlled inference latency, the systematic relationship between VLA latency and robot task success rate remains uncharacterized.

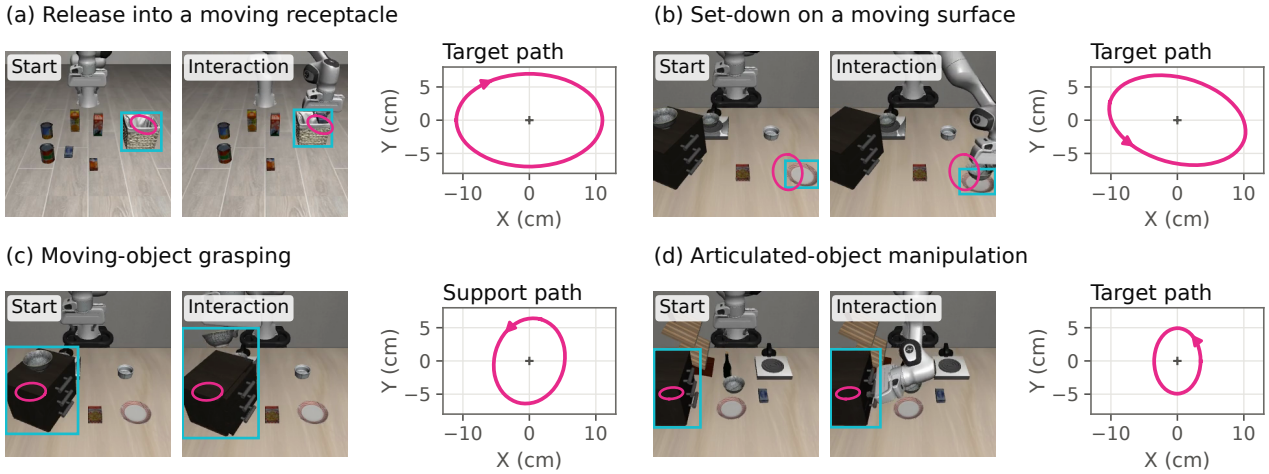


Figure 4. The four primary tasks in VLALAGBENCH cover releasing, setting down, grasping, and articulated-object manipulation. Each target, highlighted by a cyan box, moves continuously along the closed path shown in pink without colliding with background objects.

3 BENCHMARK AND EXECUTION MODES

We present VLALAGBENCH, a benchmark for systematically studying how VLA inference latency affects task success. As summarized in Figure 1, we vary five factors: (1) manipulation task type, (2) target object speed, (3) VLA policy, (4) execution mode, and (5) inference latency. We hope the insights from the systematic evaluation of VLALAGBENCH (§4) will guide future robotic system and architecture designs.

3.1 Design Principles

To ensure that performance drops reflect latency sensitivity rather than baseline policy failures or simulator artifacts, VLALAGBENCH is built around four design principles. First, evaluated policies must achieve high success rate in static scenes at zero latency so that accuracy degradation with added latency is attributable to latency. Second, the tasks should cover diverse interactions with moving targets, such as grasping a moving object or placing an object at a moving destination. Third, object moving trajectories must be precisely controllable without altering the simulator's underlying contact physics. Fourth, multiple widely adopted VLA models must provide public checkpoints on the exact same tasks to enable cross-policy comparisons.

These principles rule out most of the widely used simulated manipulation benchmarks we screened. The first principle alone excludes several benchmarks because public VLA checkpoints have limited success even in static scenes. For example, $\pi_{0.5}$ averages only 16.9% success on RoboCasa365 (Nasiriany et al., 2026) and 28.0% on RoboLab (Yang et al., 2026b). Similarly, π_0 averages 47.9% on Meta-World (Yu et al., 2020; Shukor et al., 2025) and 46.4% on the clean scenes of RoboTwin 2.0's bimanual tasks (Chen et al., 2026a), while GR00T N1.7 averages 62.3% on SimplerEnv's WidowX scenes (Li et al., 2025; NVIDIA, 2026b). Kinetix (Matthews et al., 2025) fails the second and fourth principle: its 2D physics control tasks are evaluated using small policies trained per environment rather than VLA models (Black et al., 2025c).

Thus, we build our primary benchmark on LIBERO in MuJoCo, which satisfies our design principles and allows controlled target motion while preserving realistic contact physics (Todorov et al., 2012). We use DOM in Isaac Sim for secondary cross-simulator verification (Xie et al., 2026; NVIDIA, n.d.) (Appendix E). We also verify our findings through real-robot experiments (Appendix F).

3.2 Tasks

We select four representative tasks that cover distinct ways a robot interacts with a moving target (Figure 4).

(a) Release. The robot picks up a stationary cream cheese package and drops it into a moving basket.

(b) Set-down. The robot picks up a stationary bowl and places it on a moving plate.

(c) Grasping. The robot picks up a bowl carried by a moving cabinet and places it on a stationary plate.

(d) Articulated manipulation. The robot grasps the handle of the moving cabinet's middle drawer and pulls it open.

We select these four tasks because they achieve near-perfect policy success in static scenes, show distinct robot–object interactions, and are sensitive to target motion. Appendix A explains why other LIBERO tasks were not selected.

These tasks differ in how the robot interacts with the moving target. *Release* and *set-down* deliver an object to a moving destination, whereas *grasping* and *articulated-object manipulation* require the robot to acquire a moving object or handle directly. For the moving destination tasks, release allows coarse alignment over a wide basket opening, whereas

set-down requires precise placement near the center of a narrow plate. For the moving object tasks, grasping requires intercepting the bowl and lifting it from its moving support, whereas articulated-object manipulation requires sustained contact and motion to pull the drawer open.

Each task is engineered so that driving the target along a trajectory preserves realistic contact physics upon robot interaction. For *release* and *set-down*, we kinematically update the horizontal pose of the basket and plate; because the robot drops the cream cheese from above in *release*, it avoids rigid collisions with the unyielding basket walls. For *grasping*, kinematically locking the bowl would prevent the robot from lifting it, so we instead drive the supporting cabinet beneath the bowl and let surface friction carry the bowl along the path. For *articulated-object manipulation*, we move the entire cabinet body while leaving the drawer's damped prismatic joint fully physical.

To validate our findings in a second simulator, we evaluate release and grasping in Isaac Sim using the Dynamic Object Manipulation (DOM) benchmark and the accompanying DynamicVLA policy trained on moving objects (Xie et al., 2026). In this secondary suite (Figure 22, Appendix E), the robot either drops a can into a moving box (*release*) or picks a can off a moving invisible rail carrier to place into a static box (*grasping*). However, DOM's native tasks are largely homogeneous—primarily picking up a moving object and placing it into a static receptacle—and the only model supporting this benchmark with a public checkpoint is DynamicVLA, which fails to adapt to new task setups and exhibits persistent oscillatory movements (acting as if intercepting a fast-moving target even when the object rests completely still). We therefore use LIBERO tasks as the primary benchmark and report DynamicVLA as a cross-simulator verification in Appendix E.

3.3 Motion Parameters

Each target moves at a constant speed along a closed, nearly elliptical path that avoids collisions with background objects in LIBERO's original layouts. We sweep target speeds from 0.001 m/s (near-static) to 0.2 m/s across all four tasks; Appendix B provides trajectory and initialization details.

3.4 VLA Policies

VLALAGBENCH evaluates three representative VLA policies: $\pi_{0.5}$, GR00T N1.7, and MolmoAct2, using their official public checkpoints (Physical Intelligence, 2025; NVIDIA, 2026a; Fang et al., 2026). These checkpoints achieve near-perfect success in static scenes, providing a reliable baseline for comparing latency sensitivity across policies. For cross-simulator verification on DOM in Isaac Sim, we evaluate DynamicVLA, a policy explicitly trained on moving objects on the DOM benchmark (Xie et al., 2026).

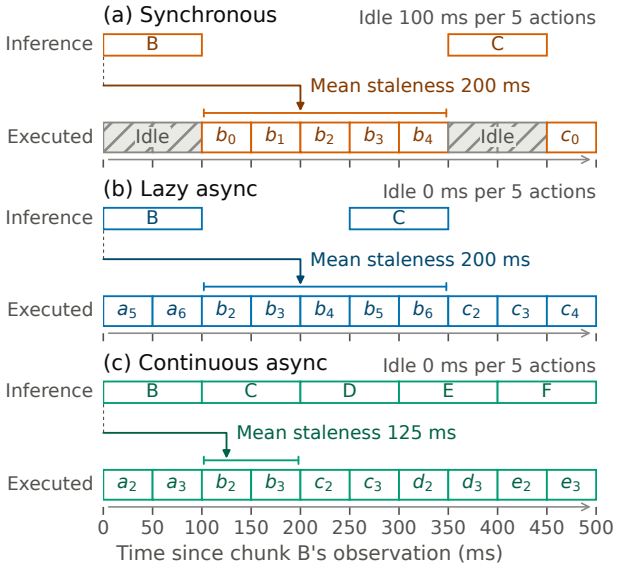


Figure 5. Synchronous and two types of asynchronous execution, assuming 100 ms inference latency and 50 ms per action.

3.5 Execution Modes

We compare synchronous execution with two representative asynchronous schedules: lazy and continuous execution (Figure 5). An execution mode governs when the robot requests a new action chunk and how it acts while waiting, creating a fundamental trade-off between robot idle time, observation staleness, and inference frequency. Across the three modes, K denotes the number of predicted actions per chunk, H the number executed per chunk, the execution horizon, and L the inference latency expressed as a number of control steps (§3.6).

Mode 1: Synchronous execution. Synchronous execution is a common deployment pattern in which inference and robot action execution interleaves (Black et al., 2025c). After executing H actions, the robot requests a new chunk and pauses for L control steps while inference completes, holding its position and last gripper command. Once the action chunk becomes available, the robot resumes execution from its first predicted action.

Pros and cons. Its advantage is its low inference request rate: one request every $H + L$ control steps. Its disadvantage is that the robot pauses during every inference call, which can disrupt tasks that require continuous motion.

Idle time. We quantify robot pauses using idle time (I), the number of control steps spent waiting for inference per H executed actions. Synchronous execution incurs $I = L$ idle steps per execution horizon (hatched in Figure 5). Once the first action chunk is available, both asynchronous modes introduced below eliminate pauses ($I = 0$), provided the available actions cover the next inference interval.

Mode 2: Lazy asynchronous execution. Asynchronous execution avoids robot pauses by running inference while the robot executes actions from an earlier chunk. Lazy execution requests inference least frequently among various asynchronous schedules, starting the next call only when L of the H scheduled actions remain so that the next chunk becomes available as the current execution horizon ends. When the new chunk arrives, the robot skips its first L actions, which correspond to the elapsed inference interval, and executes the following H actions.

Pros and cons. Its advantage is that it keeps the robot moving continuously without pauses. Its disadvantages are that (1) more frequent inference does not reduce average observation staleness relative to synchronous execution, and (2) robot motion during inference can introduce additional uncertainty for tasks requiring precise manipulation.

Observation staleness. We define observation staleness as the average time elapsed between capturing a visual observation and executing an action generated from it. As illustrated in Figure 5, synchronous execution runs actions 0 through $H-1$ of each chunk, yielding an average staleness of $(H-1)/2+L$ with an inference latency of L steps. Because lazy asynchronous execution skips the first L actions, its observation staleness is identically $(H-1)/2+L$ steps.

Mode 3: Continuous asynchronous execution. Continuous execution represents the opposite extreme in asynchronous execution, running inference as frequently as possible by starting each new call as soon as the previous one completes, while the robot continues executing previously predicted actions. At each completion, the robot switches to the new chunk, skips its first L actions, and executes the next L actions while another inference call runs, yielding an execution horizon of $H=L$ and an average observation staleness of $(3L-1)/2$ steps.

Pros and cons. Its advantage is more frequent action updates and thus a lower observation staleness than in either of the other modes when latency is shorter than their execution horizon ($L<5$). The average staleness is even lower than in zero-latency synchronous execution with $H=5$ when $L=1$. Its disadvantages are a higher inference request rate and more frequent transitions between chunks, which can introduce discontinuities in the robot's actions.

3.6 Latency Injection Mechanism

We simulate inference latency by converting it into a number of simulator control steps. For example, at a control frequency of 20 Hz, a latency of 100 ms corresponds to $L=2$ steps: synchronous execution inserts two stationary actions that hold the robot's position and last gripper command, whereas asynchronous execution continues executing available actions from the previous chunk. We extend each task's deadline by the number of inference-induced idle steps to preserve its action budget (Appendix B).

4 EVALUATION AND FINDINGS

In this section, we use VLALAGBENCH to (1) examine how inference latency affects success across tasks, (2) identify the factors leading to accuracy loss, (3) assess the trade-offs between synchronous and asynchronous execution, and (4) compare latency sensitivity across VLA policies. We present the main findings here, with the full experimental setup, complete results, failure stages, and validation in Isaac Sim with DynamicVLA (Appendix E) and on real robots (Appendix F) provided in the appendices.

4.1 Evaluation setup

On LIBERO-based tasks, we test three widely used VLA models, including $\pi_{0.5}$, GR00T N1.7, and MolmoAct2 at a 20 Hz control rate with an execution horizon of $H=5$ and their default chunk sizes of $K=10$ ($\pi_{0.5}$, MolmoAct2) or $K=16$ (GR00T). Each reported success rate is based on 200 rollouts, with a budget of 600 executed actions per episode. This increased budget, up from 220–300 actions in standard static LIBERO, reduces failures caused by insufficient time to pursue moving targets. We test inference latencies from 0 to 250 ms in 50 ms increments because (1) this range covers latencies encountered in practical VLA inference systems (Xu et al., 2026; Jiang et al., 2026), and (2) at 250 ms ($L=5$), policies with $K=10$ reach the asynchronous execution limit $K=H+L$.

4.2 Latency sensitivity across tasks

Q1. *How does inference latency affect task success across different manipulation tasks?*

Finding 1. *Inference latency can substantially reduce task success on moving targets, with the extent of degradation depending on both the task nature and target speed.*

Empirical success curves confirm that latency penalties emerge primarily when targets move and scale differently across tasks. As shown in Figure 6, an inference latency of 200 ms under synchronous execution changes success on nearly static targets (0.001 m/s) by only a few percentage points across all tasks. On moving targets, however, even moderate inference latency affects tasks differently: at a target speed of 0.1 m/s, 100 ms of latency nearly halves *set-down* success, from 71.5% to 40.5% (57% of its zero-latency success rate). In contrast, *grasping* success declines only from 86.5% to 74.0%, retaining 86% of its zero-latency success rate. We next examine the factors underlying these differences in latency sensitivity across tasks.

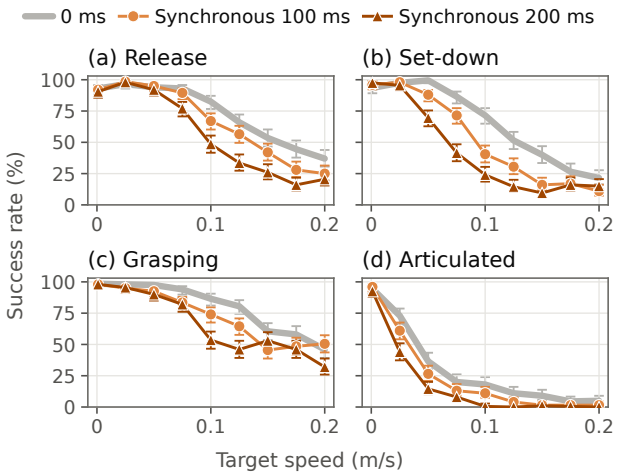


Figure 6. Task success across target speeds and inference latencies using $\pi_{0.5}$ with synchronous execution. Different tasks exhibit distinct latency sensitivities.

Q2. *What factors determine a task's sensitivity to latency?*

Finding 2. *A task's latency sensitivity depends on its requirements for action precision and continuous motion. (2.1) Tasks requiring high spatial precision suffer from stale observations, whereas (2.2) tasks requiring sustained motion suffer from robot pauses.*

(2.1) Precision. Stale observations make precise manipulation harder because the target moves between image capture and action execution, so the robot may act on a position the target no longer occupies. Figure 7 compares the precision required by different tasks: *set-down* requires placing the bowl within a narrow 3 cm zone around the plate's center, whereas *release* tolerates drops anywhere across a 12.2 cm basket opening. At a target speed of 0.1 m/s, 200 ms of additional observation staleness allows the plate to move 2 cm, a displacement equal to two-thirds of *set-down*'s tolerance for placement error, making precise placement more difficult.

(2.2) Continuous motion. Robot pauses disrupt tasks that require continuous motion, making it harder to catch up with or move alongside the target. Because VLA policies are trained via imitation learning on static demonstrations (Liu et al., 2023), they issue actions at a nearly constant velocity regardless of how fast the target moves (Figure 11, Appendix B). Synchronous execution halts the arm after every H actions, reducing $\pi_{0.5}$'s average approach speed by 34% at 200 ms latency during *release*. This reduced speed causes the arm to fall behind the moving basket before releasing, and prevents the gripper from keeping pace with the moving drawer during *articulated-object manipulation*.

As we quantify in §5.1, *grasping* is more sensitive to observation staleness, *articulated manipulation* and *release* are more sensitive to robot pauses, while *set-down* is sensitive to both.

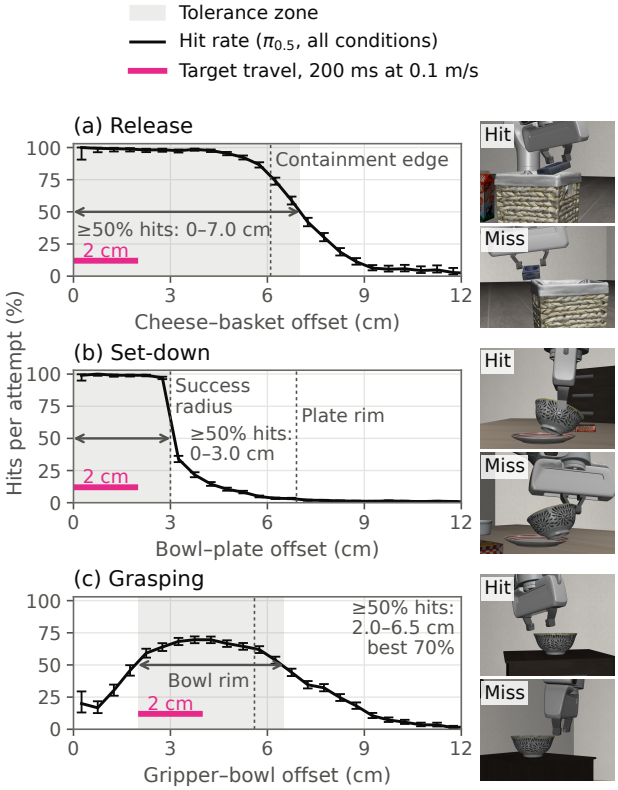


Figure 7. Hit rate of $\pi_{0.5}$'s attempts against their horizontal offset from the target center when the object leaves the gripper (*release*), first touches the surface (*set-down*), or the fingertips descend below the object's rim (*grasping*); the tolerance zone is the range of offsets where at least half of the attempts hit, and the photos show one hit inside it above one miss outside it. *Set-down* requires greater precision than the other tasks.

4.3 Synchronous versus asynchronous inference

Q3. *What are the trade-offs between synchronous and asynchronous execution as inference latency increases?*

Finding 3. *Asynchronous execution may recover task success by reducing robot pauses and observation staleness, at the cost of more frequent inference (§3.5). (3.1) Both lazy and continuous asynchronous execution keep the robot moving during inference, helping tasks that require continuous motion, such as* articulated-object manipulation. *(3.2) At short inference latencies, continuous asynchronous mode updates actions more frequently, helping tasks that require precise manipulation, such as* set-down.

(3.1) Removing pauses. For *release* and *articulated-object manipulation*, the two tasks more sensitive to robot pauses (§4.2), both asynchronous modes help by keeping the robot moving during inference. Figure 8(a,d,e,h) shows that both modes restore $\pi_{0.5}$'s success close to the zero-latency base-

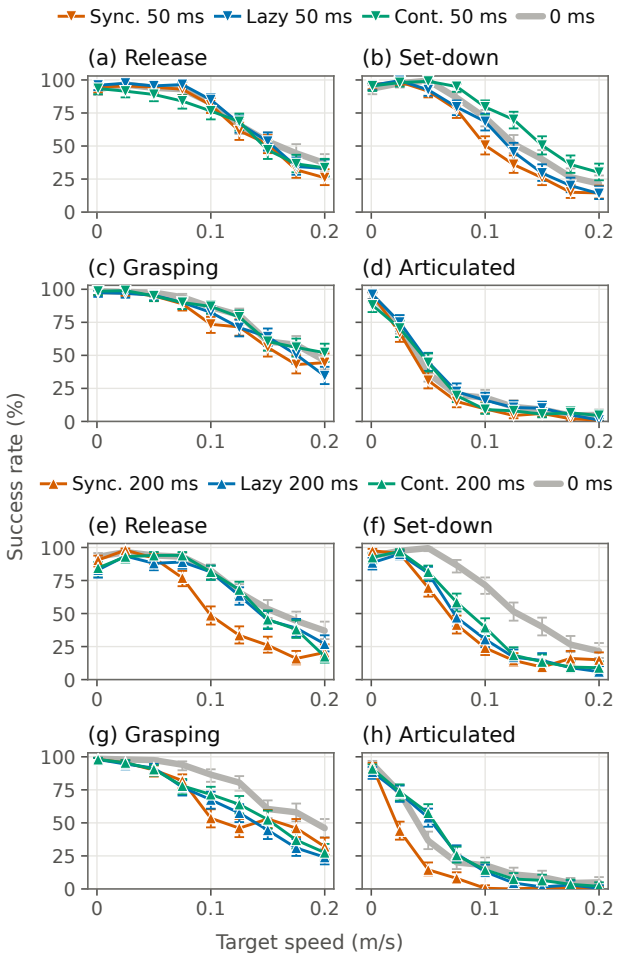


Figure 8. Task success using $\pi_{0.5}$ under synchronous and asynchronous execution at 50 and 200 ms inference latency.

line at both 50 and 200 ms. In particular, synchronous and lazy asynchronous execution have the same average observation staleness at the same inference latency (§3.5), so the recovery under lazy execution indicates that removing pauses accounts for much of the improvement.

(3.2) Reducing observation staleness. For *set-down* and *grasping*, the two tasks more sensitive to stale observations (§4.2), continuous asynchronous execution can improve success by updating actions more frequently when inference latency is sufficiently short. At 50 ms ($L = 1$), continuous asynchronous execution updates the action chunk at every control step, reducing average observation staleness below that of zero-latency synchronous execution ($H = 5$) and increasing *set-down* success at 0.125 m/s from 51.5% to 70.0% (Figure 8b). The benefit is more modest for *grasping*: continuous asynchronous outperforms synchronous and lazy execution at several target speeds, but does not consistently exceed the zero-latency baseline (Figure 8c). As inference latency approaches lazy execution's horizon ($L \to 5$), continuous asynchronous updates actions less frequently and

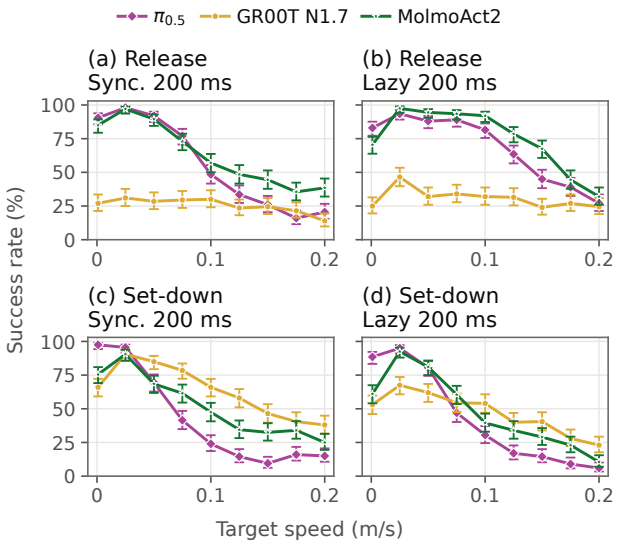


Figure 9. Task success across VLA policies at 200 ms latency. The best-performing policy depends on the task and target speed.

its observation staleness approaches that of lazy execution; at 200 ms, neither asynchronous mode consistently recovers zero-latency success on these two tasks (Figure 8(f,g)).

4.4 Latency sensitivity across VLA policies

Q4. *Do VLA models exhibit the same latency sensitivity?*

Finding 4. *VLA policies differ in their latency sensitivity, so the best-performing policy at a given inference latency depends on both the task and target speed (4.1). This variation reflects differences in (4.2) robustness to changes in initial target position and (4.3) sensitivity to observation staleness and robot pauses.*

(4.1) Different tasks and speeds favor different policies. As shown in Figure 9, $\pi_{0.5}$ achieves the highest success at the slowest target speeds on both *release* and *set-down* under 200 ms synchronous execution. At higher speeds, MolmoAct2 performs best on *release*, whereas GR00T N1.7 performs best on *set-down* (Figure 9(a,c)). The full four-task comparison is shown in Figure 14 (Appendix C).

(4.2) Per-policy robustness to initial target position. Table 1 shows accuracy of different policies given various initial target positions on nearly static targets (0.001 m/s). Although all three policies achieve roughly 97% success on the original static LIBERO layouts, their success rates differ substantially when initial positions are sampled along the motion path. For example, on *set-down*, $\pi_{0.5}$ achieves 93.5% success, compared with 74.0% for MolmoAct2 and 56.0% for GR00T N1.7.

(4.3) Per-policy sensitivity to observation staleness and robot pauses. VLA policies differ in their sensitivity to observation staleness and robot pauses, leading to different

Table 1. Success rate on nearly static targets (0.001 m/s) across policies. Changing initial target positions already substantially reduces success even with zero inference latency.

Task	$\pi_{0.5}$ (%)	GR00T N1.7 (%)	MolmoAct2 (%)
Release	93.0	52.0	83.0
Set-down	93.5	56.0	74.0
Grasping	98.5	98.0	99.5
Articulated	94.5	69.5	81.0

gains from asynchronous execution. For example, on *release* at a target speed of 0.1 m/s and an inference latency of 200 ms (Figure 9(a,b)), switching from synchronous to lazy asynchronous execution increases success from 48.5% to 81.5% for $\pi_{0.5}$ and from 57.0% to 92.0% for MolmoAct2, gains of 33 and 35 percentage points, respectively. In contrast, GR00T N1.7's success increases only from 30.0% to 32.0%, a much smaller gain. On *set-down*, the same switch can instead reduce GR00T N1.7's success (Figure 9(c,d)). We quantify these policy-specific sensitivities in §5.1.

5 System-Specific Configuration Selection with VLALagBench

In this section, we build on the preceding findings to guide VLA configuration selection for a given inference system. Specifically, we develop an empirical model that uses limited calibration data for each policy–task pair to predict task success across inference latencies, execution modes, and execution horizons (§5.1). This model reuses measurements collected on one system to select configurations that maximize predicted task success on a new system (§5.2).

5.1 An Empirical Model for Task Success Prediction

Success model intuition. *Stale observations and robot pauses caused by inference latency can make a task harder, much like increasing the target's speed. This similarity suggests a way to reuse past success measurements: we first measure how task success varies with target speed under one latency and execution mode. To estimate success under another latency and execution modes, we simply look up the success rate on the measured curve at a target speed adjusted for observation staleness and robot idle time.*

For a given policy and task, we approximate success under configuration X as $S_X(v) \approx S_0(\lambda_X v)$, where S_0 is the success curve under zero-latency synchronous execution with a fixed reference horizon. Configuration X specifies the inference latency, execution mode, and execution horizon. The factor λ_X adjusts the target speed: to predict success at speed v, we read the reference curve at speed $\lambda_X v$. In practice, we find that this adjustment is well approximated by a simple linear model of observation staleness and robot idle time:

$$\lambda_X = 1 + a\,\frac{\bar{s}_X - \bar{s}_0}{\bar{s}_0} + b\,\frac{I_X}{H_X}, \tag{1}$$

where $\bar{s}_X$ and $\bar{s}_0$ are the mean observation staleness under configuration X and the reference configuration, respectively; H_X is the execution horizon, and I_X is the number of idle steps per execution horizon. The ratio $(\bar{s}_X - \bar{s}_0)/\bar{s}_0$ measures the relative change in observation staleness, while I_X/H_X measures the number of idle control steps per executed action (§3.5). The coefficients a and b are fitted separately for each policy–task pair and quantify its sensitivity to these two factors.

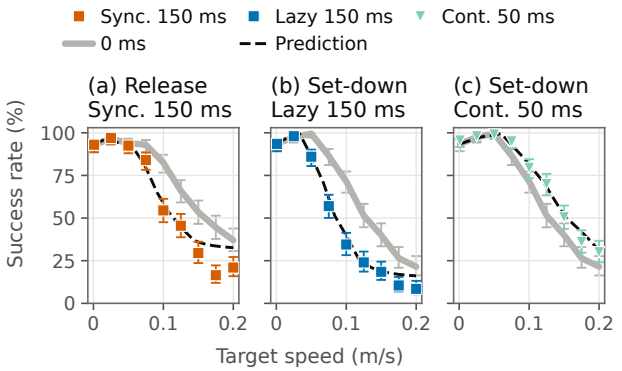


Figure 10. Model predictions track $\pi_{0.5}$'s measured success at inference latencies excluded from calibration. Dashed black curves show predictions; markers show measurements.

Calibrating the model does not require zero-latency measurements: the reference success curve can come from synchronous or asynchronous execution at a nonzero latency. For a reference configuration R, we use $S_X(v) \approx S_R((\lambda_X/\lambda_R)v)$, where the ratio adjusts target speed relative to R. To estimate the sensitivity to robot pauses (coefficient b), the calibration data must include at least one synchronous configuration with nonzero latency.

Our model predicts $\pi_{0.5}$'s task success with a root-mean-square (RMS) error of 5.7% across the four tasks and three execution modes, with representative predictions shown in Figure 10. For this evaluation, we calibrate on zero-latency synchronous execution and all three modes at 100 and 200 ms, then predict success at 50, 150, and 250 ms. Even with substantially less calibration data—one complete success curve plus one point from each of two other configurations—the median RMS error on the remaining measurements is 7.7% across the evaluated calibration choices for $\pi_{0.5}$ (Table 5 in Appendix D). At 50 ms latency, we also evaluate asynchronous execution with horizons of $H = 2$, 3, and 4, which lie between those of continuous ($H = 1$) and lazy ($H = 5$) execution. For these configurations, our model achieves RMS prediction errors of 3.3% on *set-down* and 2.8% on *release*. Appendix D reports the full calibration and horizon results, along with separately

Table 2. Fitted sensitivities to observation staleness (a) and robot idle time (b) across tasks and policies. †GR00T N1.7 is an outlier with strong sensitivity to initial target positions.

	$\pi_{0.5}$		MolmoAct2		GR00T N1.7†	
Task	a	b	a	b	a	b
Release	0.04	0.59	−0.09	0.80	2.48	1.31
Set-down	0.33	0.32	0.31	0.09	−1.00	3.65
Grasping	0.23	0.07	0.23	−0.06	0.26	−0.37
Articulated	−0.03	1.41	0.09	1.37	−0.02	1.19

calibrated predictions for other VLA policies.

We fit the model for each policy–task pair (Table 2) to show their sensitivity to staleness and pauses. For $\pi_{0.5}$, *grasping* is more sensitive to observation staleness, *release* and *articulated-object manipulation* to robot pauses, and *set-down* to both. MolmoAct2 follows the same pattern except on *set-down*, where it is more sensitive to staleness. GR00T N1.7 is an outlier: its fitted coefficients are negative for observation staleness on *set-down* and robot idle time on *grasping* (Table 2). This behavior may partly reflect its sensitivity to initial target positions: changing these positions from LIBERO's original layouts reduces its zero-latency success to 52% on *release* and 56% on *set-down*, even with nearly static targets (Table 1). As a consequence, near-static *set-down* targets accidentally yield even lower success than the fastest targets even at zero latency (Figures 9 and 15). These behaviors limit how well target-speed scaling captures its latency sensitivity.

5.2 System-Oriented Algorithm Space Exploration

Achieving the highest task success rate on a given inference system requires choosing the right combination of execution mode and execution horizon. The GPU, network, software runtime, and resource-sharing arrangement jointly determine the system's inference latency and available throughput per robot, so a configuration that performs well on one system may not be the best choice for another. Exploring this configuration space experimentally is expensive because each candidate requires repeated task executions to estimate its success rate. *Our goal is therefore to reuse past deployment data to select configurations that maximize task success on new systems.*

We apply the prediction model from §5.1 to four representative system deployment scenarios (Table 3). These comprise an onboard GPU with relatively high inference latency (System A), a dedicated edge server with low latency (System B), a shared server with low latency but limited inference throughput per robot (System C), and a remote cloud server with high network latency (System D). GPU inference measurements (Table 6) and network latency assumptions are detailed in Appendix D (Su, 2026; Jiang et al., 2026). We calibrate the model for $\pi_{0.5}$ on *set-down* using one complete success curve (continuous execution at 100 ms) and one point from each of two additional curves (synchronous and continuous execution at 200 ms). We then predict success at a target speed of 0.075 m/s across feasible execution modes and horizons, selecting the configuration with the highest predicted success for each system. The selected configurations match the measured best configurations on all four systems, with prediction errors ranging from 2.5% to 4.6% on Systems B, C, and D. System A selects continuous execution at 100 ms, whose success curve is used for calibration.

Table 3. Using measurements from System A, our model predicts the best configurations for Systems B–D. Each system favors a different execution mode and horizon. *Calibration configuration.

System	GPU / network	Latency (ms)	Rate limit (req/s)	Predicted Mode, H	Success (%) Pred./Meas.
A	Jetson AGX Thor	72.8	Uncapped	Cont., 2	84.5*/84.5
B	RTX 4090, WiFi 7	37.8	20.0	Cont., 1	92.3/95.0
C	H100, 5G	42.9	5.0	Async., 4	84.5/82.0
D	H100, slow cloud	217.9	Uncapped	Lazy, 5	46.1/41.5

Different systems favor different execution modes and horizons, even for the same VLA policy and task (Table 3). System A favors continuous execution ($H = 2$) to keep the robot moving while requesting inference as frequently as its onboard GPU allows. System B's lower latency supports continuous execution with a shorter horizon ($H = 1$), providing fresher observations for precise placement. System C's limited per-robot throughput instead favors a longer asynchronous horizon ($H = 4$), a mode between lazy and continuous asynchronous execution, that sustains robot motion within the request budget. System D's high network latency removes the staleness advantage of continuous execution, and the model favors lazy asynchronous execution ($H = 5$) for this task. Appendix D provides the complete success predictions and recommended configurations across inference latencies and per-robot throughput limits.

6 CONCLUSION AND FUTURE WORK

VLALAGBENCH represents a foundational step toward understanding how VLA inference latency affects task success. We identify the factors that shape latency sensitivity and compare execution modes and VLA policies across tasks and target speeds. We further develop an empirical model that predicts task success and guides execution-mode selection for a given inference system. We hope these findings will provide useful guidance for future robotics algorithm–system co-design solutions.

REFERENCES

Akinola, I., Xu, J., Song, S., and Allen, P. K. Dynamic grasping with reachability and motion awareness. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9422–9429, 2021. doi: 10.1109/IROS51168.2021.9636057. URL https://arxiv.org/abs/2103.10562.

Arachchige, N. R., Chen, Z., Jung, W., Shin, W. C., Bansal, R., Barroso, P., He, Y. H., Lin, Y. C., Joffe, B., Kousik, S., and Xu, D. SAIL: Faster-than-demonstration execution of imitation learning policies. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 721–749. PMLR, 2025. URL https://proceedings.mlr.press/v305/arachchige25a.html.

Black, K., Brown, N., Darpinian, J., Dhabalia, K., Driess, D., Esmail, A., Equi, M. R., Finn, C., Fusai, N., Galliker, M. Y., Ghosh, D., Groom, L., Hausman, K., Ichter, B., Jakubczak, S., Jones, T., Ke, L., LeBlanc, D., Levine, S., Li-Bell, A., Mothukuri, M., Nair, S., Pertsch, K., Ren, A. Z., Shi, L. X., Smith, L., Springenberg, J. T., Stachowicz, K., Tanner, J., Vuong, Q., Walke, H., Walling, A., Wang, H., Yu, L., and Zhilinsky, U. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M. R., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., Jakubczak, S., Jones, T., Ke, L., Levine, S., Li-Bell, A., Mothukuri, M., Nair, S., Pertsch, K., Shi, L. X., Smith, L., Tanner, J., Vuong, Q., Walling, A., Wang, H., and Zhilinsky, U. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025b. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Black, K., Galliker, M. Y., and Levine, S. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, pp. 33383–33407, 2025c. doi: 10.52202/085713-1122. URL https://papers.nips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

Chen, T., Chen, Z., Chen, B., Cai, Z., Liu, Y., Li, Z., Liang, Q., Lin, X., Ge, Y., Gu, Z., Deng, W., Guo, Y., Nian, T., Xie, X., Chen, Q., Su, K., Xu, T., Liu, G., Hu, M., Gao, H.-a., Wang, K., Liang, Z., Qin, Y., Yang, X., Luo, P., and Mu, Y. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pp. 13673–13699. PMLR, 2026a. URL https://proceedings.mlr.press/v306/chen26j.html.

Chen, Y., Zhang, W., and Li, X. Reflex: Enabling fast and predictive vision-language-action models for reaction-critical manipulation. *arXiv preprint arXiv:2608.14379*, 2026b. URL https://arxiv.org/html/2608.14379v1.

Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B. C., and Song, S. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://roboticsproceedings.org/rss19/p026.html.

Fang, H., Duan, J., Clay, D., Wang, S., Liu, S., Huang, W., Fan, X., Tsai, W.-C., Chen, S., Wang, Y. R., Xing, S., Cho, J., Park, J. S., Eftekhar, A., Sushko, P., Farley, K., Wadhwa, A., Harrison, C., Han, W., Lee, Y.-C., VanderBilt, E., Hendrix, R., Ellawela, S., Ngoo, L., Chai, J., Ren, Z., Farhadi, A., Fox, D., and Krishna, R. MolmoAct2: Action reasoning models for real-world deployment. In *Conference on Robot Learning*, 2026. URL https://arxiv.org/abs/2605.02881. To appear.

Guo, L., Xue, Z., Xu, Z., and Xu, H. DemoSpeedup: Accelerating visuomotor policies via entropy-guided demonstration acceleration. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 599–609. PMLR, 2025. URL https://proceedings.mlr.press/v305/guo25a.html.

Jiang, W., Clemons, J., Sankaralingam, K., and Kozyrakis, C. How fast can I run my VLA? Demystifying VLA inference performance with VLA-Perf. *arXiv preprint arXiv:2602.18397*, 2026. URL https://arxiv.org/html/2602.18397v1.

Jing, D., Nie, J., Zhang, T., Liu, J., Yao, H., Lu, Z., and Ding, M. TempoVLA: Learning speed-controllable vision-language-action policies. *arXiv preprint arXiv:2606.06491*, 2026. URL https://arxiv.org/abs/2606.06491.

Kim, M. J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E. P., Sanketi, P. R., Vuong, Q., Kollar, T., Burchfiel, B., Tedrake, R., Sadigh, D., Levine, S., Liang, P., and Finn, C. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine*

Learning Research, pp. 2679–2713. PMLR, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Lee, J., Duan, J., Fang, H., Deng, Y., Liu, S., Li, B., Fang, B., Zhang, J., Wang, Y. R., Lee, S., Han, W., Pumacay, W., Wu, A., Hendrix, R., Farley, K., Vanderbilt, E., Farhadi, A., Fox, D., and Krishna, R. MolmoAct: Action reasoning models that can reason in space. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2585–2592, 2026. doi: 10.1109/ICRA57385.2026.11696546. URL https://ieeexplore.ieee.org/document/11696546.

Li, X., Hsu, K., Gu, J., Mees, O., Pertsch, K., Walke, H. R., Fu, C., Lunawat, I., Sieh, I., Kirmani, S., Levine, S., Wu, J., Finn, C., Su, H., Vuong, Q., and Xiao, T. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

Liu, B., Zhu, Y., Gao, C., Feng, Y., Liu, Q., Zhu, Y., and Stone, P. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Ma, Y., Zhou, Y., Yang, Y., Wang, T., and Fan, H. Running VLAs at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025. URL https://arxiv.org/abs/2510.26742.

Matthews, M., Beukman, M., Lu, C., and Foerster, J. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zCxGCdzreM.

Nasiriany, S., Maddukuri, A., Zhang, L., Parikh, A., Lo, A., Joshi, A., Mandlekar, A., and Zhu, Y. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.050. URL https://www.roboticsproceedings.org/rss20/p050.html.

Nasiriany, S., Nasiriany, S., Maddukuri, A., and Zhu, Y. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tQJYKwc3n4.

Niu, J., Gu, K., Zhao, Y., Liang, S., Wang, T., Hu, X., Wang, Y., and Li, H. Realtime-VLA FLASH: Speculative inference framework for diffusion-based VLAs. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2605.13778. To appear.

NVIDIA. GR00T N1.7 LIBERO: Model card and checkpoints. Hugging Face model repository, 2026a. URL https://huggingface.co/nvidia/GR00T-N1.7-LIBERO. Evaluated revision: 2ea293aa20ba7cf5bbf3ba17a5fbcb1a01cbfe21. LIBERO results: Isaac-GR00T examples/LIBERO/README.md, commit 1a1837f20538b7d7e21f977a11a5aee14f99803c.

NVIDIA. GR00T N1.7 SimplerEnv evaluation. GitHub repository, 2026b. URL https://github.com/NVIDIA/Isaac-GR00T/blob/1a1837f20538b7d7e21f977a11a5aee14f99803c/examples/SimplerEnv/README.md. examples/SimplerEnv/README.md, commit 1a1837f20538b7d7e21f977a11a5aee14f99803c.

NVIDIA. Isaac Sim. Robotics simulation software and documentation, n.d. URL https://docs.isaacsim.omniverse.nvidia.com/latest/index.html. Accessed October 19, 2026.

NVIDIA, Bjorck, J., Castañeda, F., Cherniadev, N., Da, X., Ding, R., Fan, L. J., Fang, Y., Fox, D., Hu, F., Huang, S., Jang, J., Jiang, Z., Kautz, J., Kundalia, K., Lao, L., Li, Z., Lin, Z., Lin, K., Liu, G., Llontop, E., Magne, L., Mandlekar, A., Narayan, A., Nasiriany, S., Reed, S., Tan, Y. L., Wang, G., Wang, Z., Wang, J., Wang, Q., Xiang, J., Xie, Y., Xu, Y., Xu, Z., Ye, S., Yu, Z., Zhang, A., Zhang, H., Zhao, Y., Zheng, R., and Zhu, Y. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. URL https://arxiv.org/abs/2503.14734.

Physical Intelligence. openpi: LIBERO benchmark example and results. GitHub repository, 2025. URL https://github.com/Physical-Intelligence/openpi/blob/2d70d966582e711128ad8358d8dbf23d2cc3d658/examples/libero/README.md. examples/libero/README.md, commit 2d70d966582e711128ad8358d8dbf23d2cc3d658.

Shukor, M., Aubakirova, D., Capuano, F., Kooijmans, P., Palma, S., Zouitine, A., Aractingi, M., Pascal, C., Russi, M., Marafioti, A., Alibert, S., Cord, M., Wolf, T., and Cadene, R. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025. URL https://arxiv.org/abs/2506.01844.

Su, L. Execution-state capsules: Graph-bound execution-state checkpoint and restore for low-latency,

small-batch, on-device physical-AI serving. *arXiv preprint arXiv:2606.20537*, 2026. doi: 10.48550/arXiv.2606.20537. URL https://arxiv.org/abs/2606.20537.

Tang, J., Sun, Y., Zhao, Y., Yang, S., Lin, Y., Zhang, Z., Hou, J., Lu, Y., Liu, Z., and Han, S. VLASH: Real-time VLAs via future-state-aware asynchronous inference. In *2026 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. URL https://arxiv.org/abs/2512.01031.

TheRobotStudio. SO-ARM100: Standard open arm 100 (SO-100 and SO-101). GitHub repository, 2025. URL https://github.com/TheRobotStudio/SO-ARM100.

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. URL https://homes.cs.washington.edu/~todorov/papers/TodorovIROS12.pdf.

Xie, H., Wen, B., Zheng, J., Chen, Z., Hong, F., Diao, H., and Liu, Z. DynamicVLA: A vision-language-action model for dynamic object manipulation. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2601.22153. To appear.

Xu, B., Tan, H., Jiang, W., He, B., Wu, M., Zeng, J., and Heinrich, M. VLA-HWBench: Where the time goes when VLA models run on edge hardware. *arXiv preprint arXiv:2610.34717*, 2026. URL https://arxiv.org/abs/2610.34717.

Yang, C., Hu, Y., Ma, Y., Yang, Y., Tan, J., and Fan, H. Realtime-VLA V2: Learning to run VLAs fast, smooth, and accurate. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 4501–4509, 2026a. URL https://openaccess.thecvf.com/content/CVPR2026W/GigaBrainChallenge/html/Yang_Realtime-VLA_V2_Learning_to_Run_VLAs_Fast_Smooth_and_Accurate_CVPRW_2026_paper.html.

Yang, X., Dagli, R., Zook, A., Hadfield, H., Goyal, A., Birchfield, S., Ramos, F., and Tremblay, J. RoboLab: A high-fidelity simulation benchmark for analysis of task generalist policies. In *Proceedings of Robotics: Science and Systems*, 2026b. doi: 10.15607/RSS.2026.XXII.096. URL https://www.roboticsproceedings.org/rss22/p096.html.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

Zhang, Y., Song, C., Singh, V., Li, X., Ye, K., Hu, Z., Pu, Z., Yin, Y., and Chaudhary, V. Overcoming dynamics-blindness: Training-free pace-and-path correction for VLA models. *arXiv preprint arXiv:2605.11459*, 2026. URL https://arxiv.org/html/2605.11459v1.

Zhao, T. Z., Kumar, V., Levine, S., and Finn, C. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://roboticsproceedings.org/rss19/p016.html.

Zhou, X., Xu, Y., Tie, G., Chen, Y., Zhang, G., Chu, D., Zhou, P., and Sun, L. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025. URL https://arxiv.org/abs/2510.03827.

AI USAGE DISCLOSURE

We used AI coding assistants (Claude Code and Codex) to write code, run data analysis, plot figures, search related work, check grammar, and draft some parts of the paper. The authors directed and reviewed all AI-assisted work and take full responsibility for the paper's content.

APPENDIX CONTENTS

These appendices provide supporting details for the benchmark design, evaluation, and prediction model.

Appendix A. Task selection within LIBERO p. 15

Selection criteria and how the other LIBERO tasks relate to the four chosen interactions.

Appendix B. Evaluation and measurement details p. 15

Task mappings, policy checkpoints, target paths and motion, episode design, statistical reporting, and definitions of attempts, tolerance zones, and hand speed.

Appendix C. Full success curves and recovery p. 16

Complete results across target speeds, inference latencies, execution modes, and VLA policies.

Appendix D. Model validation and calibration p. 22

Held-out prediction accuracy and its noise floor, the reference curve, limited-data calibration, deployment predictions, and where speed scaling departs from the measurements.

Appendix E. DynamicVLA in Isaac Sim: a second simulator and policy p. 26

Cross-simulator experiments on release and grasping with a policy trained on moving objects.

Appendix F. Real-robot experiments p. 28

Experiments with MolmoAct2 on an SO-101 arm across grasping and release tasks, target speeds, inference latencies, and execution modes.

Appendix G. Failure stages p. 30

Stage pass rates showing where latency costs success in each task.

Appendix H. Additional related work p. 31

Related approaches to asynchronous inference, policy execution speed, and latency compensation.

A TASK SELECTION WITHIN LIBERO

We selected the four tasks from LIBERO's four standard suites by three criteria: the evaluated checkpoints were trained on the task's suite and succeed in nearly all of its static episodes, each task adds an interaction the others lack, and moving the target lowers success enough for latency to matter. LIBERO-90 fails the first criterion: neither $\pi_{0.5}$ nor GR00T N1.7 was trained on it, and from its official initial states they succeed in 28.3% and 16.8% of static episodes (GR00T with its LIBERO-10 checkpoint). Within the four standard suites, most tasks repeat one of the four interactions or combine two subgoals. All ten LIBERO-Object tasks put a grocery item into the basket (*release*), and all ten LIBERO-Spatial tasks put the black bowl on the plate (*set-down* when the plate moves, *grasping* when the bowl's support moves). Nine of the ten LIBERO-10 tasks combine two subgoals, such as putting two items into the basket or putting a bowl into a drawer and closing it. LIBERO-Goal's tasks set objects down, put the cream cheese into a bowl (*release*), open a drawer or turn on the stove (*articulated-object manipulation*), or open a drawer and then place the bowl inside; the remaining task pushes the plate, so the robot itself moves the target.

B EVALUATION AND MEASUREMENT DETAILS

Targets follow fixed closed paths while the robot, the cameras, the language instructions, the success checks, and MuJoCo's contact model stay as in LIBERO; this section also defines the attempts, tolerance zones, and hand speed behind Figures 7 and 11. *Release* uses LIBERO-Object task 1, whose basket containment region is 12.2 cm wide; *set-down* and *grasping* use LIBERO-Spatial task 9, whose success check requires the bowl's center within 3 cm of the plate's; and *articulated-object manipulation* uses LIBERO-Goal task 0. GR00T uses suite-specific checkpoints (NVIDIA, 2026a), and MolmoAct2 uses one checkpoint for all suites (Fang et al., 2026).

Each target moves at constant speed along one closed cubic spline through 12 waypoints on an ellipse; the paths span about 22×14 cm for *release*, 21×14 cm for *set-down*, 11×13 cm for *grasping*, and 7×10 cm for *articulated-object manipulation*. We evaluate every task at 0.001 m/s and from 0.025 to 0.2 m/s in 0.025 m/s steps. The grid stops at 0.2 m/s, past the speed at which $\pi_{0.5}$'s zero-latency success falls to 50% on every task (0.04–0.19 m/s); at 0.2 m/s it succeeds in 5.0–46.0% of episodes, against 93.0–98.5% near static. We keep the background objects where LIBERO's original layouts place them, since perturbing initial object states can sharply reduce policy success (Zhou et al., 2025); in *set-down* the stationary cabinet holding the bowl starts at the same position in every layout, and in *grasping* the bowl starts at the center of its moving support. Each condition's 200 episodes cover LIBERO's 50 official initial states, each with two starting points on the path and both travel directions.

Task deadlines. Each LIBERO episode has a budget of 600 executed policy actions, increased from 220–300 in standard static LIBERO to allow more time to pursue moving targets. An episode is considered a failure if the task remains incomplete when this budget is exhausted. Stationary actions inserted during inference waits do not consume this budget: we extend the deadline by one control step for each such idle step while the target continues moving. This preserves the same executed-action budget across inference latencies and execution modes.

A driver writes each target's planar path at every physics substep. The basket and the plate are free bodies whose planar position and velocity are written while their height and rotation stay physical; the bowl's support moves on added planar slide joints whose position and velocity follow the path; and the drawer's cabinet, a jointless LIBERO fixture, is moved by position alone while the drawer keeps its damped slide joint. No gravity, mass, friction, or solver setting changes, so MuJoCo resolves every contact with its unmodified contact model.

Following OpenPI's protocol (Physical Intelligence, 2025), each episode begins with 10 warm-up steps in which the robot and the target stay still. In asynchronous modes, the robot waits only for the first chunk and executes it from its first action. Synchronous and lazy execution coincide at zero latency; the policy comparison in Appendix C pools their zero-latency runs where both exist, while the success curves and Table 1 use synchronous runs. Intervals are 95% Wilson confidence intervals.

A tolerance zone is the range of horizontal offsets from the target center within which at least half of the attempts hit. We pool attempts from all speeds, latencies, and modes, retries included, into 0.5 cm bins of offset; a bin with at least 30 attempts belongs to the zone if at least half of them hit.

Hand speed is the hand's path length per second while it closes in on the target, over every qualifying interval after warm-up, retries and inference waits included. For *release* and *set-down*, an interval qualifies while both fingers hold the object within 15 cm horizontally of the receptacle or surface and the hand is at or above the receptacle's rim or the surface; for *grasping* and *articulated-object manipulation*, it qualifies while the empty hand is within 10 cm of the object or handle without touching it. Figure 11 plots the median episode hand speed at each target speed; the 34% reduction in Section 4.2 compares the medians of these values over target speeds.

Attempts are detected from the MuJoCo state traces,

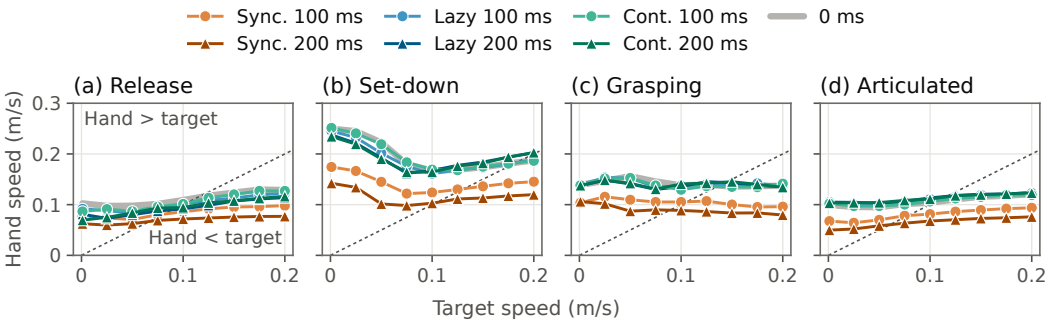


Figure 11. $\pi_{0.5}$ moves its hand at a nearly constant pace regardless of target speed, which asynchronous execution preserves and synchronous pauses reduce. Dotted lines mark hand speed equal to target speed.

recorded once per 50 ms control step, where two bodies are in contact when their contact distance is ≤ 0. A *confirmed pickup* starts the first uninterrupted run of at least three samples with both fingers in contact with the object in which the object is at least once 2 cm above its post-warm-up height. A *release attempt* begins a run of at least two samples without finger contact, after contact by both fingers within the preceding three samples, with the object lifted at least 2 cm and the gripper commanded open; its offset is measured at the last contact sample, and it hits if the episode succeeds without later contact. A *set-down attempt* begins at each onset of object–surface contact, with gaps shorter than three samples merged, and hits if the task completes during that contact. A *grasp attempt* is the first sample with the lowest fingertip below the object's rim after at least three samples above it, with the hand within 15 cm horizontally of the object, no later than the start of the first confirmed pickup or the first sample with the object more than 5 cm below its post-warm-up height; it hits if a confirmed pickup starts at or after it, before the next attempt, and within 40 samples.

C FULL SUCCESS CURVES AND RECOVERY

Figures 12, 13, 14, 15, and 16 give every tested speed, latency, execution mode, and policy behind Section 4. On MolmoAct2, zero-latency success is higher at 0.025 m/s than at 0.001 m/s on *release* (91.0% vs. 83.0%) and *set-down* (100% vs. 74.0%; Figure 16). In its near-static set-down failures, MolmoAct2 tends to leave the bowl on the plate's rim and then stop, whereas a plate moving at 0.025 m/s drags the bowl onto itself.

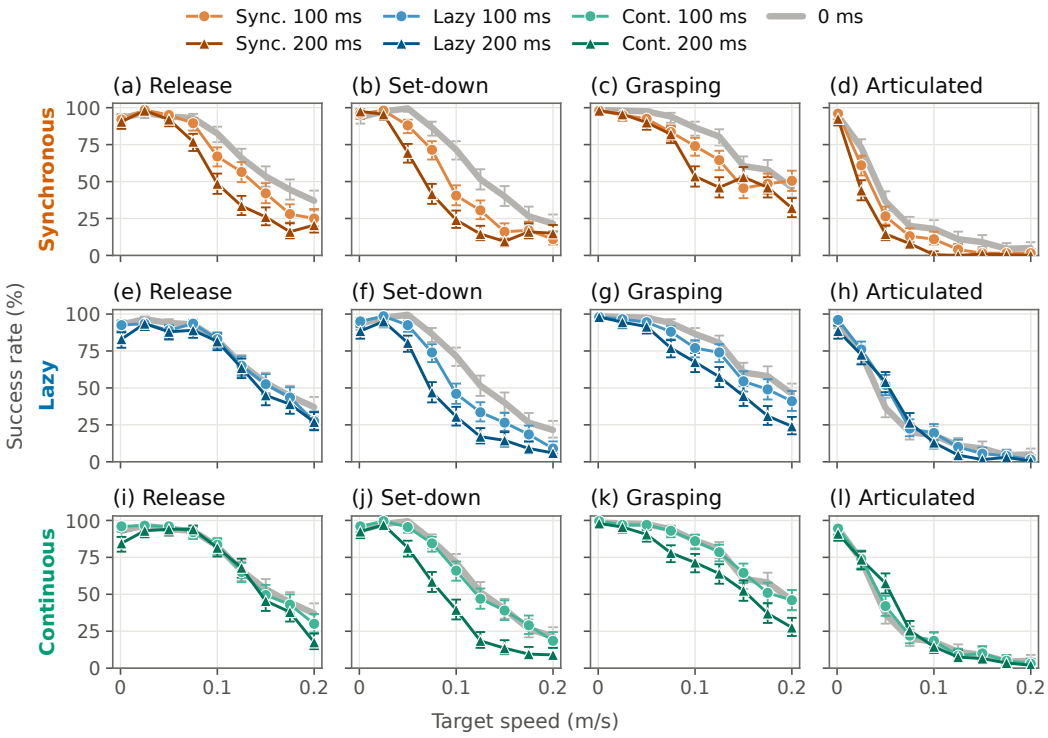


Figure 12. At 100 and 200 ms latencies, lazy execution eliminates most synchronous losses on release and articulated-object manipulation, whereas at 200 ms set-down loses success on moving targets in all three modes. Curves show $\pi_{0.5}$'s measured success at the calibration latencies.

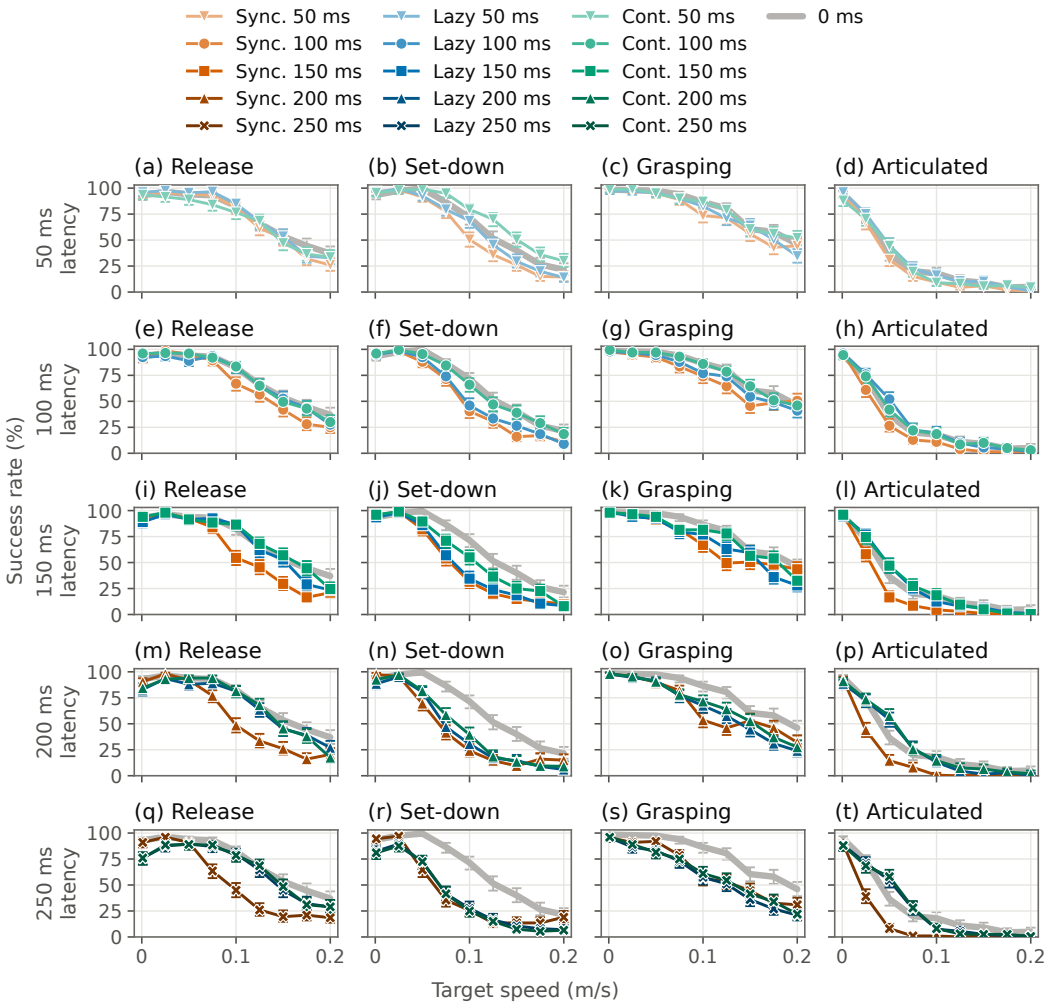


Figure 13. Continuous execution outperforms zero-latency execution on set-down for $\pi_{0.5}$ at 50 ms before converging toward lazy execution as latency approaches lazy execution's horizon, while 250 ms asynchronous execution loses success even near static.

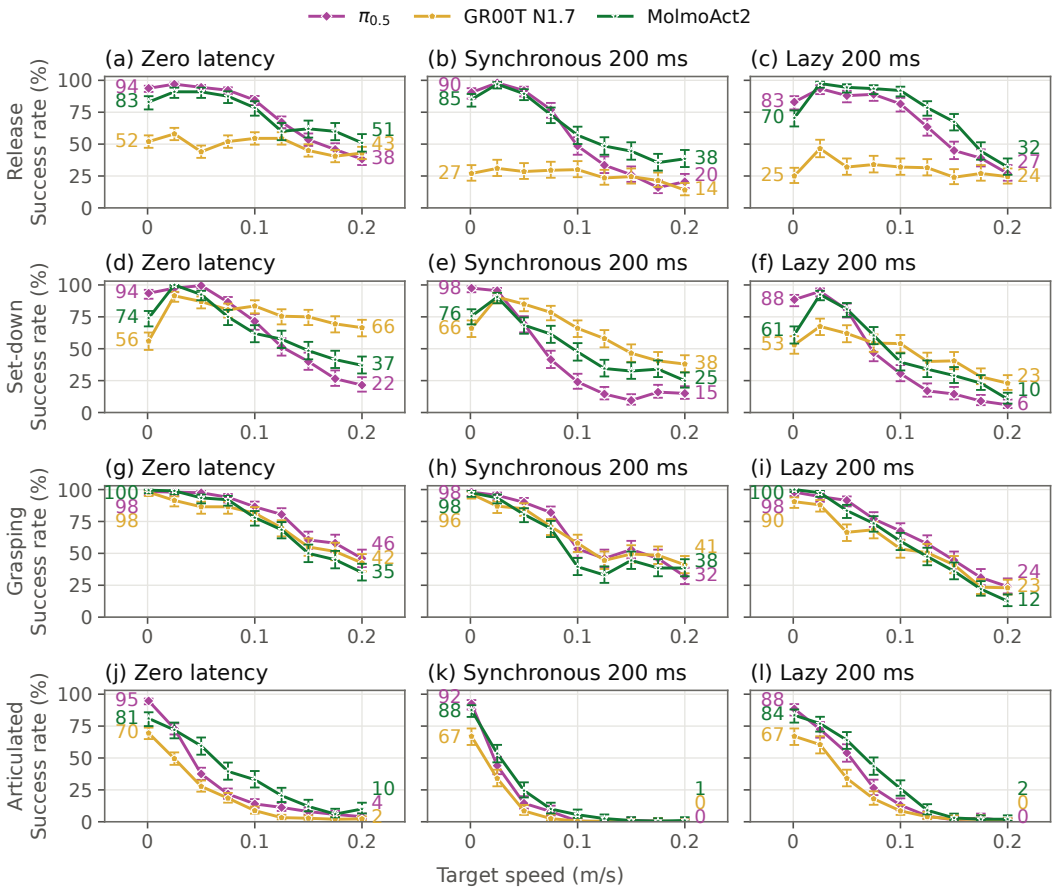


Figure 14. Policy rankings invert across tasks and speeds: GR00T N1.7 trails near static on three tasks but surpasses $\pi_{0.5}$ on fast set-down targets, while MolmoAct2 leads on moving articulated targets.

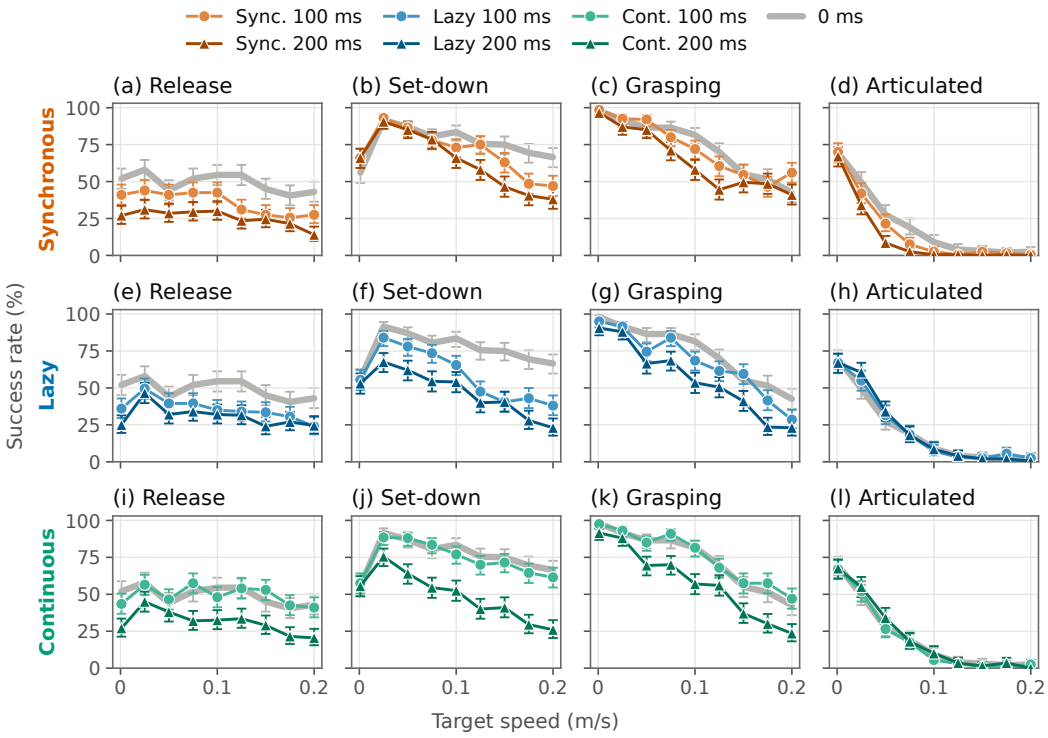


Figure 15. On articulated-object manipulation at 0.05 m/s, lazy execution at 200 ms raises GR00T N1.7's success from 8.5% under synchronous execution to 34.0% (27.5% at zero latency), but on release at 0.1 m/s only from 30.0% to 32.0%; on grasping at 200 ms, synchronous execution beats lazy execution at six of eight moving-target speeds.

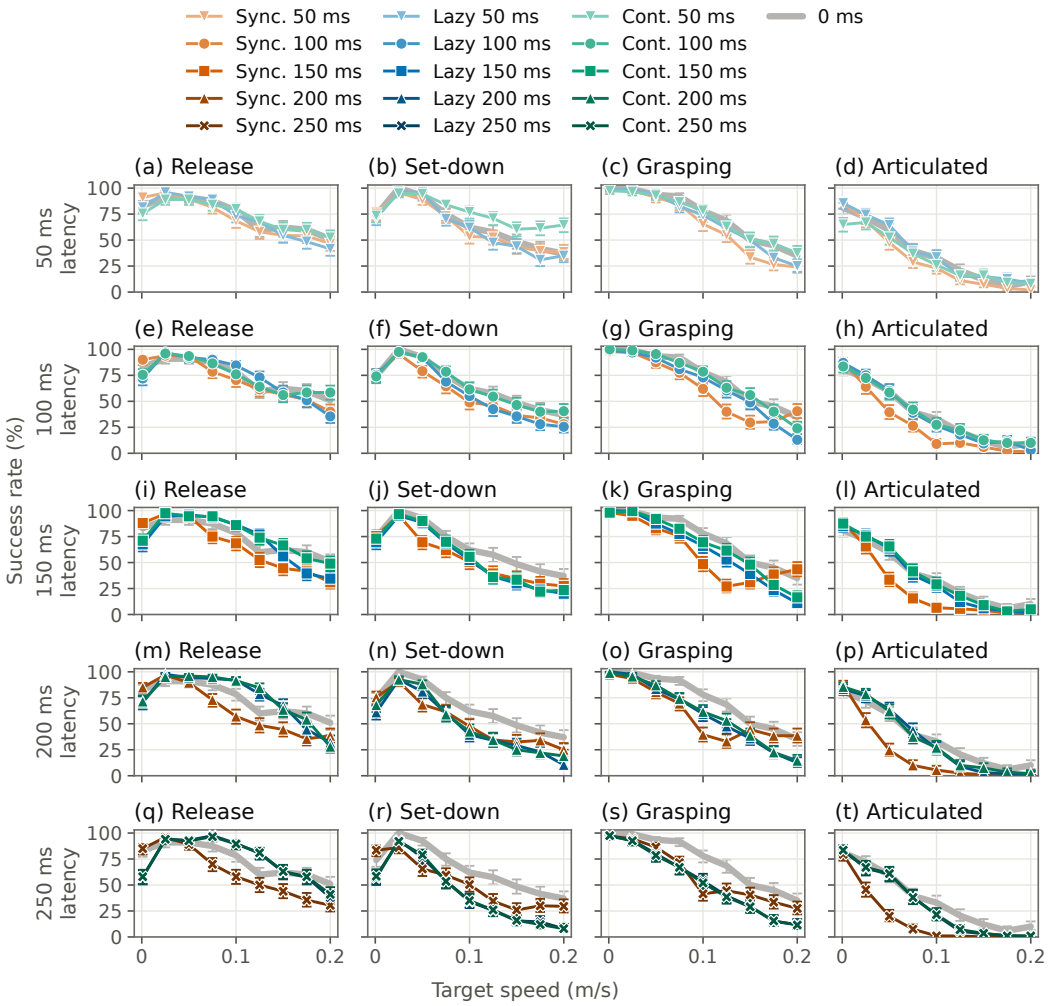


Figure 16. On MolmoAct2, continuous execution at 50 ms exceeds zero-latency success on set-down from 0.075 m/s, by 9.0–27.5 points, and zero-latency success is lower at 0.001 m/s than at 0.025 m/s on release and set-down.

D MODEL VALIDATION AND CALIBRATION

Fitted on 0, 100, and 200 ms runs, the model predicts the held-out 50, 150, and 250 ms curves with an RMS error of 5.7 points for $\pi_{0.5}$ and 7.8 points for MolmoAct2 (Table 4, Figures 17 and 18). Sampling noise alone would make an exactly right model miss these measurements by 4.0 points RMS for $\pi_{0.5}$ (3.9 for MolmoAct2), combining the held-out points' own binomial noise, 2.6 points (2.8), with that of the measured reference curve each prediction reads.

Each prediction reads the reference curve, the measured success curve of the reference configuration, at the target speed scaled by λ from Equation 1. Between measured target speeds the curve interpolates the measured success rates linearly, with no monotonicity imposed; below the slowest measured speed it keeps that speed's success rate; above the fastest it follows a log-logistic curve fitted by maximum likelihood to the reference's own points and scaled to meet the measured success at that speed. Apart from this tail, which depends on the reference curve alone, the model's only free parameters are a and b, fitted by binomial maximum likelihood on the calibration conditions. For $\pi_{0.5}$, the model also predicts asynchronous execution at 50 ms with intermediate horizons of $H = 2$, 3, and 4, which no calibration includes, with RMS errors of 3.3 points on *set-down* and 2.8 points on *release*.

Our model can be calibrated without zero-latency data from one complete curve plus one point on each of two other curves. Because $S_Y(v) \approx S_X(v\,\lambda_Y/\lambda_X)$ for any two settings X and Y, the complete curve X, read as measured, fixes the curve shape, and the two points constrain the ratios λ_Y/λ_X, from which a and b are fitted by maximizing the binomial likelihood times a weak Gaussian prior over a bounded grid. One of the three curves must be synchronous at a nonzero latency, because only it separates idle time from staleness.

We score every such calibration on the 0–250 ms grid for $\pi_{0.5}$ and MolmoAct2: each complete curve at 50 to 250 ms in any mode, with one point in each of the other two execution modes, both at the grid speed nearest that curve's midpoint speed divided by 1.4, where its measured success first falls halfway between its highest and lowest value (Table 5). On $\pi_{0.5}$, the median RMS error over all remaining points is 7.7 points, 80% of calibrations fall within 4.5–14.4 points, and the unseen zero-latency curve is recovered within a median of 6.8 points. MolmoAct2's median is 11.1 points. GR00T N1.7, measured only at 0, 100, and 200 ms, has a median of 11.6 points. For $\pi_{0.5}$, measuring both points on the sloped part of the curve, near the midpoint speed divided by 1.4, matters: over every pair of grid speeds, its median error rises to 9.4 points.

Table 6 lists the 99th-percentile latency of a single in-process request at bf16 with openpi, run in PyTorch, and with FlashRT (Su, 2026). For networked deployments, we add approximately 5 ms of network latency over WiFi 7, 25 ms over 5G, and 200 ms over a wired link to a slow cloud region, following the estimates in VLA-Perf (Jiang et al., 2026). System A's latency is instead the median that VLA-HWBench reports for $\pi_{0.5}$ under FlashRT at 16-bit precision on an NVIDIA Jetson AGX Thor, with our workload of 10 actions and two camera images (Xu et al., 2026).

From the calibration of Section 5.2, the continuous 100 ms curve plus synchronous and continuous points at 200 ms, the model predicts $\pi_{0.5}$'s set-down success at every speed, latency, and mode (Figure 19); FlashRT sets only the latency, and every success rate comes from our openpi JAX runs. Each latency takes effect at the next whole 50 ms control step: 37.8 and 42.9 ms act as 50 ms, 72.8 ms as 100 ms, and 217.9 ms as 250 ms. Across the 133 measured points outside the calibration, the model errs by 4.2 points RMS against 2.6 points of sampling noise in the measured points (Table 7). Its predicted best mode matches the measured best in 36 of 45 latency-by-speed cells, counting ties, including every latency at 0.075 m/s; at 250 ms, lazy and continuous execution run the same schedule and are predicted equal. All nine misses fall at 150–250 ms; in each, the model ranks its choice less than 3 points above the measured best mode, and the measured best mode beats the choice by at most 7.5 points, except at 250 ms and 0.2 m/s (12.0).

Figure 20 gives, from the same calibration, the best schedule each granted inference rate allows at each latency.

Speed scaling departs from the measurements in two places. First, on $\pi_{0.5}$'s fastest targets, synchronous execution keeps more success than both asynchronous modes: on *grasping* at 0.2 m/s at every latency from 100 to 250 ms (200 ms: 32.0% against 24.0% lazy and 27.5% continuous) and on *set-down* at 0.175–0.2 m/s at 200 and 250 ms (250 ms, 0.2 m/s: 19.0% against 7.0% and 6.5%; Figure 13). The model gives synchronous execution lazy execution's staleness plus idle time, so on these falling parts of the reference curve it predicts synchronous execution 0.2–1.6 points below lazy execution. Second, at 250 ms, lazy and continuous execution lose success near static (0.001 m/s) on *release* (93.0% to 76.0% in both modes) and *set-down* (93.5% to 82.5% and 81.0%), where after the first chunk they execute actions 5–9 of each chunk, while synchronous execution, executing actions 0–4, stays within two points (91.0%, 94.5%); the model moves these near-static predictions by less than 0.2 points. Points read from the extrapolated reference curve beyond 0.2 m/s, about a fifth of the held-out points, carry 33% of $\pi_{0.5}$'s held-out squared error and 37% of MolmoAct2's, whose held-out error gathers at the three fastest speeds (0.15–0.2 m/s: 58%, against 44% for $\pi_{0.5}$).

Table 4. Calibrating Equation 1 on 0, 100, and 200 ms runs predicts held-out 50, 150, and 250 ms curves with an RMS error of 5.7 percentage points on $\pi_{0.5}$ and 7.8 points on MolmoAct2.

RMS error (points)	Release	Set-down	Grasping	Articulated	All
$\pi_{0.5}$					
Synchronous	7.1	2.8	5.5	3.8	5.1
Lazy	6.3	5.3	5.7	5.2	5.6
Continuous	6.5	5.7	6.7	6.1	6.3
Every mode	6.7	4.8	6.0	5.1	5.7
Point sampling noise	2.7	2.6	2.8	2.2	2.6
MolmoAct2					
Synchronous	8.0	5.0	9.8	3.1	7.0
Lazy	11.2	10.2	4.2	4.9	8.2
Continuous	8.3	11.3	5.6	6.0	8.1
Every mode	9.3	9.2	7.0	4.8	7.8
Point sampling noise	2.9	3.0	2.8	2.5	2.8

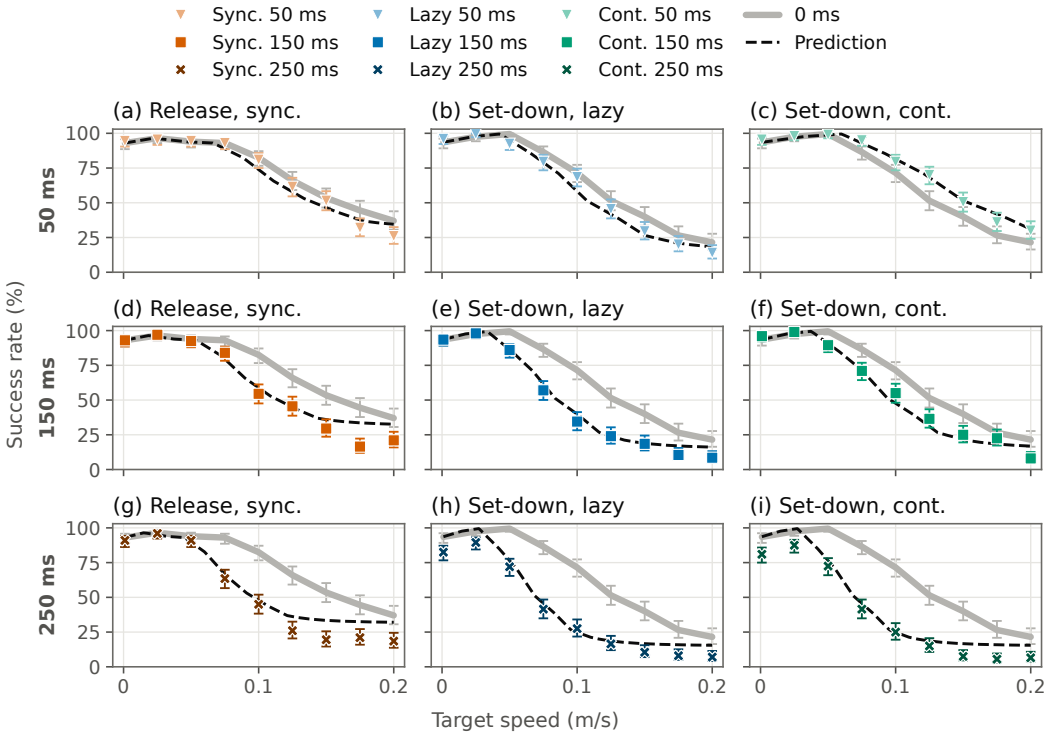


Figure 17. Speed-scaling predictions fitted on 0, 100, and 200 ms runs track $\pi_{0.5}$'s held-out 50, 150, and 250 ms curves in three representative settings; over every task and mode, the held-out error grows with latency (4.2, 5.5, and 7.0 points RMS). Dashed black curves show model predictions beside measured markers.

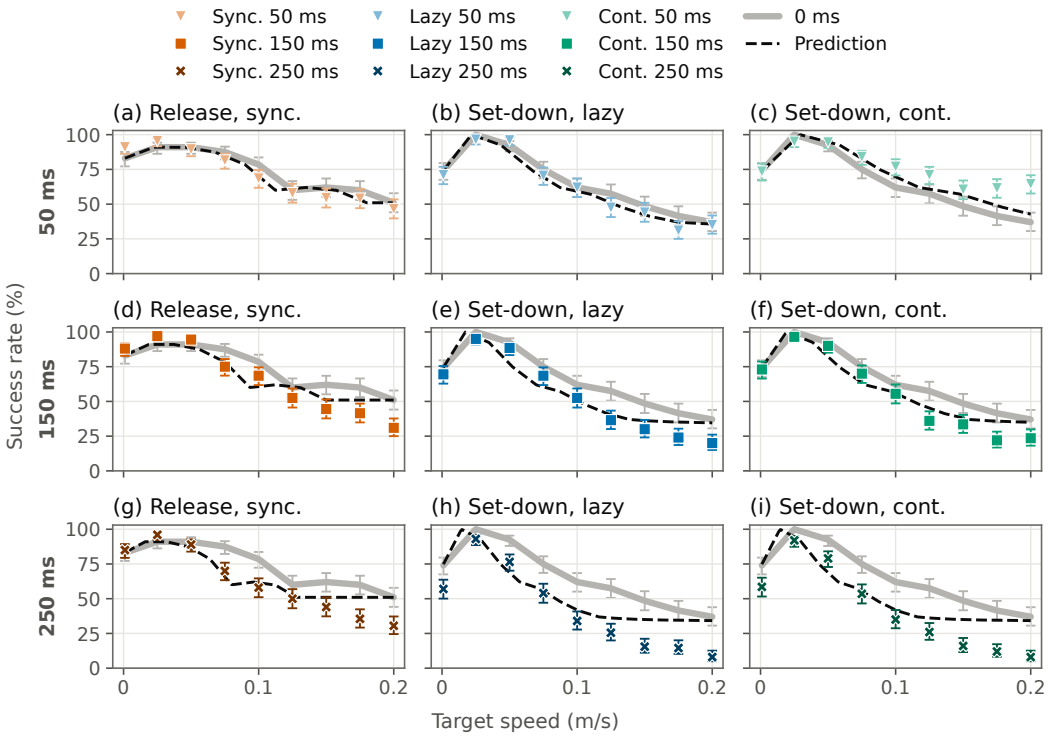


Figure 18. Held-out predictions on MolmoAct2 capture speed scaling across latencies and carry the zero-latency dip of near-static set-down, with larger residuals at 250 ms. Dashed black curves show predictions from 0, 100, and 200 ms fits beside measured markers.

Table 5. One complete curve, with one point on each of two other curves, predicts the remaining curves, including the unmeasured zero-latency curve.

RMS error (points)	Release	Set-down	Grasping	Articulated	All
$\pi_{0.5}$: one complete curve and two points, no zero-latency data					
Median (80% of calibrations)	8.5 (5.4–18.4)	6.8 (4.2–15.0)	9.1 (6.2–13.0)	5.4 (4.2–10.9)	7.7 (4.5–14.4)
Its zero-latency curve, median	5.9	7.2	8.0	5.8	6.8
Fit at 0, 100, 200 ms; held-out latencies	6.7	4.8	6.0	5.1	5.7
MolmoAct2: one complete curve and two points, no zero-latency data					
Median (80% of calibrations)	12.3 (8.6–19.0)	12.2 (8.1–24.3)	10.8 (8.2–14.0)	7.4 (4.8–19.2)	11.1 (6.5–19.4)
Its zero-latency curve, median	10.9	10.3	8.6	6.3	8.6
Fit at 0, 100, 200 ms; held-out latencies	9.3	9.2	7.0	4.8	7.8

Table 6. FlashRT cuts $\pi_{0.5}$'s 99th-percentile inference latency (ms) by 46–73% relative to openpi on every GPU.

GPU	openpi	FlashRT
RTX 3090	131.6	66.3
RTX 4090	60.6	32.8
RTX 5090	56.8	27.4
H100	66.3	17.9

Table 7. The model predicts $\pi_{0.5}$'s set-down success in every execution mode within about twice the sampling noise (2.6 percentage points RMS), slightly overestimating asynchronous execution.

Mode	Points	RMS error	Mean error
Synchronous	53	3.4	−0.6
Lazy	45	4.5	+2.6
Continuous	35	5.0	+2.8
All	133	4.2	+1.4

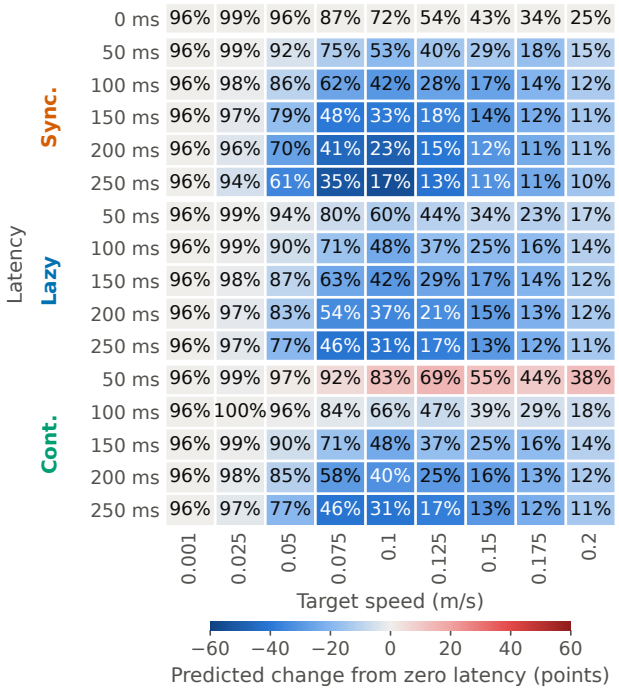


Figure 19. On set-down, the model predicts the highest $\pi_{0.5}$ success for continuous execution at nearly every moving-target speed up to 200 ms latency and for lazy and continuous execution, tied, at 250 ms; synchronous execution does not lead at any speed from 0.025 to 0.2 m/s. Cells show predicted success across target speeds and latencies from calibrating on one complete curve (continuous execution at 100 ms) plus one point on each of two other curves (synchronous and continuous execution at 200 ms).

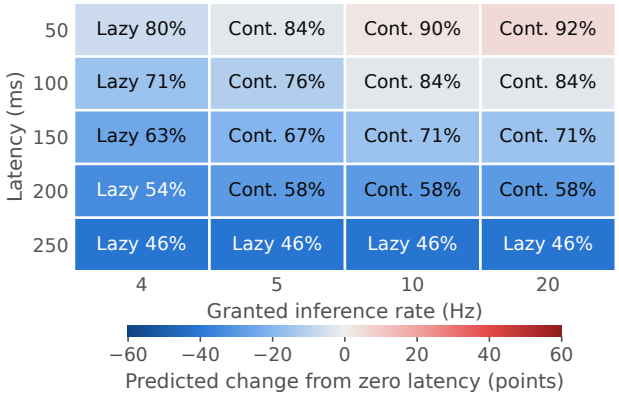


Figure 20. On set-down at 0.075 m/s, continuous execution gives $\pi_{0.5}$ the highest predicted success up to 200 ms at 5 Hz or more, and lazy execution at 4 Hz and at 250 ms. Under a rate below one inference per latency, continuous execution requests once per granted period and switches to each chunk on arrival.

E DYNAMICVLA IN ISAAC SIM: A SECOND SIMULATOR AND POLICY

We test the official DOM checkpoint of DynamicVLA (Xie et al., 2026) in Isaac Sim on *release* and *grasping* at a 25 Hz control rate with an execution horizon of $H = 14$ and a chunk size of $K = 20$. A driver writes the planar position and velocity of each moving body before every control step: the box in *release* and, in *grasping*, an unrendered rail carrier that moves the can through contact alone (Figure 22). We run synchronous execution at 0, 120, and 240 ms and lazy and continuous execution at 120 and 240 ms, where 240 ms ($L = 6$) is the longest latency satisfying $K \geq H + L$. Each setting runs 200 episodes at each target speed. Each episode has a budget of 750 executed policy actions, increased from DOM's native 300 to allow more time to pursue moving targets, and DOM's drop checks end it early as a failure if the can or the box falls off the table.

Finding 1 holds. Inference latency substantially reduces task success on moving targets, to an extent that depends on the task and target speed. Synchronous execution at 240 ms does not lower near-static success on either task (release 98.5% to 100%, grasping 95.0% to 99.5%), but it lowers the target speed at which success falls to 50% from 0.21 to 0.13 m/s on *release* and from 0.080 to 0.047 m/s on *grasping* (Figure 23a,b).

Finding 2 holds. A task's latency sensitivity depends on its requirements for action precision and continuous motion. Figure 21 compares the precision the two tasks require: *grasping* requires closing the fingers within 3 cm of the can's center, whereas *release* tolerates drops within 12.5 cm of the box's center. At a target speed of 0.1 m/s, 240 ms of additional observation staleness allows the target to move 2.4 cm, four-fifths of *grasping*'s tolerance zone.

Finding 3 holds, except on slow release targets. Asynchronous execution recovers task success, and continuous asynchronous execution recovers the most. At 240 ms, lazy execution keeps *grasping* within 7.5 points of zero latency at every speed, but raises the 50%-success speed of *release* only from synchronous execution's 0.13 to 0.15 m/s, against 0.21 m/s at zero latency (Figure 24c,d). Continuous execution at 120 ms acts on fresher observations than zero-latency synchronous execution (160 ms against 260 ms of average staleness) and raises success on fast targets in both tasks (Figure 24a,b). The exception is *release* up to 0.1 m/s, where continuous execution falls below lazy execution at both latencies: near static, it succeeds in 73.0% of episodes at 120 ms and 90.5% at 240 ms, against 99.5% for lazy execution (Figure 24a,c). At 120 ms, it drops the can early: in 36 of its 54 near-static failures, the gripper is commanded open while the can is still more than 3 cm up and more than 20 cm from the box.

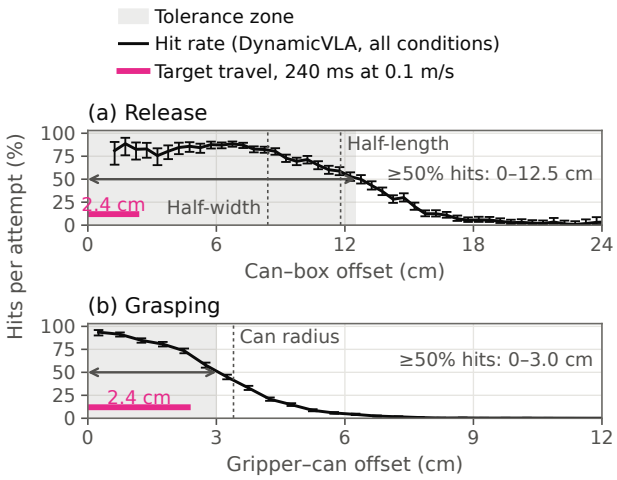


Figure 21. Hit rate of DynamicVLA's attempts against their horizontal offset from the target center when the can leaves the gripper (*release*) or the fingertips descend below the can's top (*grasping*). *Grasping* requires greater precision than *release*.

The model of Section 5.1 fits DynamicVLA's *grasping* within about twice the sampling noise and its *release* less closely, mostly where continuous execution drops the can early. Fitted on every curve at nonzero latency, Equation 1 gives *grasping* $a = 0.16$ (relative to DynamicVLA's 260 ms reference staleness) and $b = 1.37$, with an in-sample RMS error of 5.0 points against 2.4 points of sampling noise in the measured points. Pauses are thus the main cost of latency for DynamicVLA's *grasping*, unlike for $\pi_{0.5}$'s, whose b is 0.07 (Table 2). On *release*, the fit gives $a = 0.38$ and $b = 0.38$, with an RMS error of 7.9 points against 2.6, most of it from continuous execution at 120 ms on targets up to 0.1 m/s, where measured success falls 20–26 points below the fit.

DynamicVLA also moves differently from $\pi_{0.5}$: it swings its hand wide even toward a near-static can. In near-static episodes, its hand path to a 2 cm lift of the can is more than 2.5 times the straight-line distance in 86.5–100% of lifting episodes in every setting except continuous execution at 120 ms (28% on *release*, 24% on *grasping*), swinging a median 40 cm to the side of the can; $\pi_{0.5}$'s hand path does so in at most 0.5% of near-static release episodes.

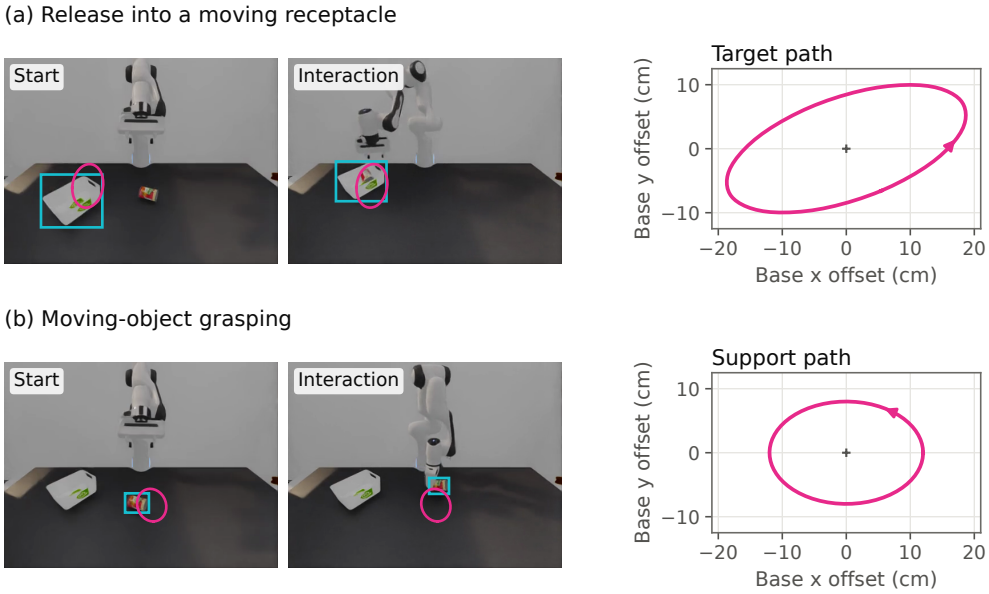


Figure 22. Our Isaac Sim verification replicates release and grasping with DynamicVLA along closed trajectories in robot-base coordinates. A cyan box marks the moving object and pink traces its path; an unrendered rail carrier moves the grasping can.

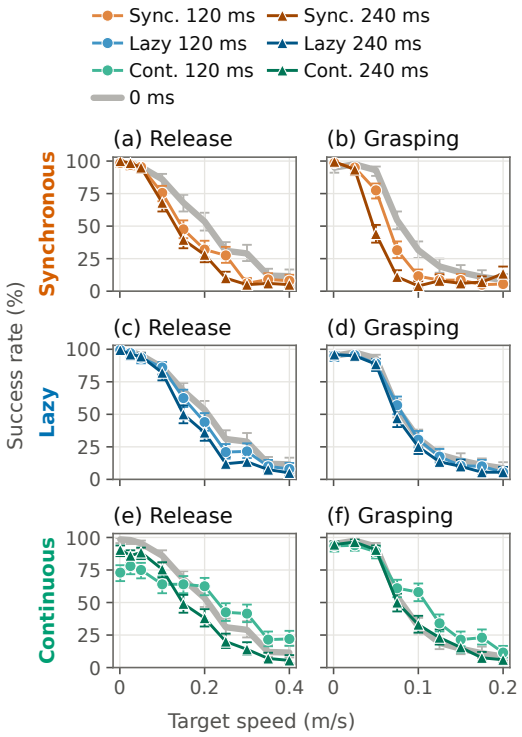


Figure 23. On DynamicVLA in Isaac Sim, near-static grasping stays at 93.5% or higher in every setting, synchronous latency moves grasping success to lower speeds, and asynchronous grasping never falls more than 7.5 points below its zero-latency curve, while 120 ms continuous execution exceeds it from 0.075 m/s.

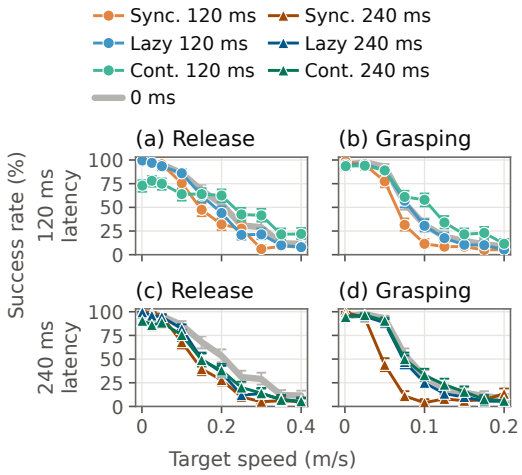


Figure 24. Lazy execution keeps DynamicVLA's grasping within 7.5 points of its zero-latency curve at 240 ms, while 120 ms continuous execution raises success on fast targets and lowers it on slow release targets.

F REAL-ROBOT EXPERIMENTS

We test the official checkpoint of MolmoAct2 for the SO-100 and SO-101 arms (Fang et al., 2026) on an SO-101 arm (TheRobotStudio, 2025) with a front camera and a wrist camera at a 30 Hz control rate with an execution horizon of $H = 16$ and its default chunk size of $K = 30$. We do not test $\pi_{0.5}$ or GR00T N1.7 on the robot because neither has an official checkpoint for this arm, and, as in simulation (§3.4), we evaluate only official public checkpoints rather than policies we fine-tune ourselves.

We select four tasks that pair two interaction types with a small and a big target: a 10×6 cm mini tote bag 7 cm tall and a 20×7 cm case 8 cm tall as the objects to grasp, and a bowl 15 cm in diameter and 5 cm tall and a plate 25 cm in diameter and 2 cm tall as the receptacles. In **(a) grasping small**, the robot picks up the moving tote bag and places it in a stationary bowl. In **(b) grasping big**, the robot picks up the moving case and places it on a stationary plate. In **(c) release small** and **(d) release big**, the robot picks up the stationary tote bag and drops it into the moving bowl or onto the moving plate (Figure 25).

Every moving target travels once, in one direction, along the same straight 30 cm path, which ends as the target leaves the camera's view. A rollout succeeds if, by then, the tote bag or case comes to rest in the bowl or on the plate and stays there without tipping over. For *grasping*, the object starts moving once the robot has executed its first action chunk. For *release*, the receptacle starts moving once the robot has lifted the tote bag off the table. A motor pulls the moving object through a transparent wire at a constant 0.02 or 0.05 m/s, and only one object moves at a time. Each rollout starts with the arm and the objects reset to the same positions.

We first check the setup with static targets, then move the targets. Under synchronous execution, we place the target of every task at the start, middle, and end of its path and test every latency. Every static setting succeeds in all five rollouts, which confirms that the path, the object placement, and the tasks themselves work before the targets move. We then move the targets at 0.02 and 0.05 m/s under synchronous execution on every task, and under lazy and continuous asynchronous execution on *release small*, the harder release task, to measure how much asynchronous execution recovers. We set the latencies from MolmoAct2's measured median inference latency of 209 ms: holding each action chunk until L control steps after its observation, we test 267, 367, and 467 ms ($L = 8$, 11, and 14), all above the median so that nearly every chunk takes effect at the set latency, with continuous execution at 267 ms ($H = L = 8$). Each reported result is based on five rollouts.

Finding 1 holds. Inference latency substantially reduces task success on moving targets, to an extent that depends on the task and target speed. Unlike static targets, which never fail, moving targets lose success: at 0.02 m/s, increasing latency from 267 to 467 ms reduces *grasping small* success from three to zero of five rollouts and *release small* success only from five to three (Figure 26(a,c)).

Finding 2 holds. A task's latency sensitivity depends on its requirements for action precision and continuous motion. At 0.02 m/s and 467 ms, *grasping big* and *release big* succeed in one and four of five rollouts, whereas *grasping small* and *release small*, which require more precision, succeed in zero and three (Figure 26). A faster target requires faster sustained motion: at 267 ms, *release small* succeeds in all five rollouts at 0.02 m/s but in two at 0.05 m/s (Figure 26c).

Finding 3 holds. Asynchronous execution recovers task success, and continuous asynchronous execution recovers the most. On *release small* at 0.05 m/s and 267 ms, continuous execution succeeds in all five rollouts, lazy execution in four, and synchronous execution in two (Figure 27a).

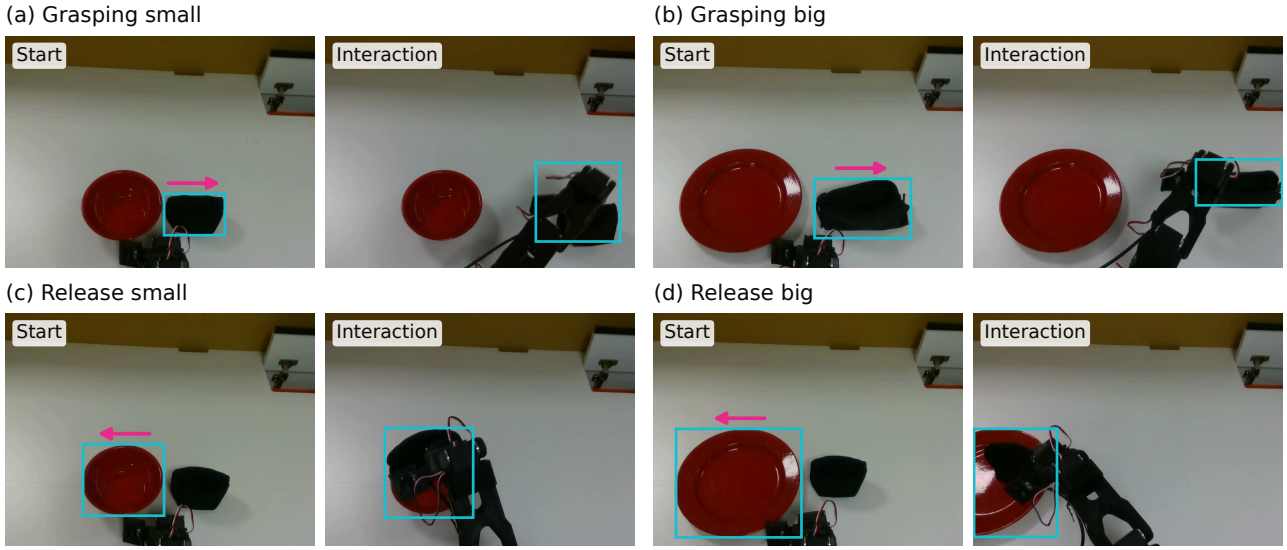


Figure 25. The four real-robot tasks pair grasping and release with a small and a big target, shown from the front camera at the start of a rollout and as the robot interacts with the moving target. Each moving target, highlighted by a cyan box, travels once along a straight path in the direction of the pink arrow.

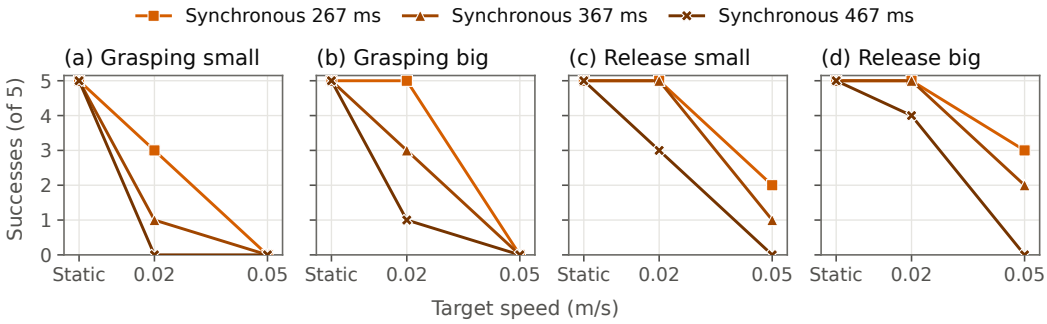


Figure 26. Task success on the SO-101 arm across target speeds and inference latencies using MolmoAct2 with synchronous execution. Static targets succeed in every rollout, while moving targets lose success as latency increases.

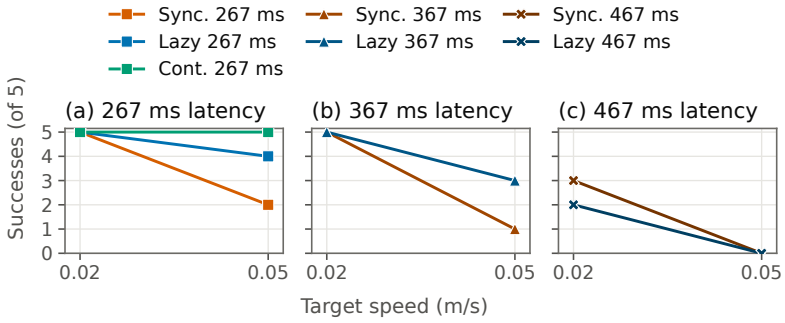


Figure 27. Task success on *release small* using MolmoAct2 under synchronous and asynchronous execution at each inference latency. Continuous execution, and lazy execution at shorter latencies, recover success on the faster target.

G FAILURE STAGES

Latency costs success at the stages where the robot interacts with the moving target, while the stages before and away from it hold (Table 8). Each task is split into the stages an episode passes on the way to success, and each rate counts the episodes passing a stage among those that reached it. In *release*, the robot picks up the static object and lets go of it over the moving receptacle; pickup means both fingers hold the object and it rises at least 2 cm, and release means both fingers then lose contact with it. In *set-down*, placement means the object rests on the moving surface with stable contact for 0.2 s. In *grasping*, the robot picks up the moving object and then places it on a static surface. In *articulated-object manipulation*, the hand first touches the handle of the moving drawer, and opening means the drawer moves at least 1 cm from its starting position. On *release*, 200 ms synchronous latency lowers success after release from 80% to 58%, while pickup and release stay at or above 92%. On *grasping*, the loss is at pickup of the moving object (85% to 73%); once the object is held, placing it on the static surface succeeds about as often as without latency. Asynchronous execution recovers these stages on *articulated-object manipulation* and partly on *release*, but little on *set-down* and *grasping*.

Table 8. Stage pass rates of $\pi_{0.5}$ (%), each among the episodes reaching that stage, over all nine target speeds; ▷ marks stages at the moving target.

	Synchronous		Lazy	Continuous
Task / stage	0 ms	200 ms	200 ms	200 ms
Release				
Pickup	100	100	100	100
Release	92	96	98	96
▷ Success after release	80	58	69	72
Set-down				
▷ Placement	90	65	67	70
▷ Success after placement	72	66	65	67
Grasping				
▷ Pickup	85	73	73	76
Success after pickup	94	91	89	90
Articulated-object manipulation				
Handle contact	97	98	91	92
▷ Opening $\geq$ 1 cm	66	48	63	62
▷ Success after opening	47	38	52	54

H ADDITIONAL RELATED WORK

Several model and system papers handle inference latency or execution speed within their own methods. On the systems side, SmolVLA introduces an asynchronous inference stack that overlaps chunk prediction with execution and compares it with synchronous execution on three real-world tasks (Shukor et al., 2025), and VLASH conditions asynchronous inference on robot states rolled forward with the previous action chunk (Tang et al., 2026). On the policy-pacing side, SAIL executes imitation policies faster than their demonstrations and schedules actions around system latency (Arachchige et al., 2025), DemoSpeedup accelerates demonstrations by entropy-guided downsampling without modeling inference delay (Guo et al., 2025), and TempoVLA conditions $\pi_{0.5}$ on a commanded speed (Jing et al., 2026). Akinola et al. (2021) predict object motion for dynamic grasping because computation delays the robot's response.