FASTER AND BETTER? BENCHMARK BUGS AND DESIGN LIMITATIONS DISTORT THE EVALUATION OF VISION-LANGUAGE-ACTION ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Simulated manipulation benchmarks are the standard tool for evaluating vision-language-action (VLA) policies and the acceleration methods that reduce their inference latency for on-robot deployment. On these benchmarks, we observe that some training-free acceleration methods, which approximate the baseline policy's computation, achieve higher measured success rates than the baseline itself. Success rates alone cannot establish whether such gains come from better task execution or from evaluation flaws. We therefore investigate two kinds of benchmark flaws behind these gains: bugs, where the implementation does not match the intended task or evaluation protocol, and design limitations, where success criteria and simulation settings do not fully capture how acceleration affects task execution. Starting from tasks with anomalous gains, we localize root causes by plotting object trajectories against checker acceptance regions, and classify the resulting bugs into task consistency, initialization, and reproducibility. Extending this audit to seven benchmarks, including RoboTwin, LIBERO-Plus, and VLABench, we identify 22 bugs of these types and 4 design limitations. For the latter, we revise permissive success checkers, correct unrealistic object masses, and add a motion-aware score that favors smoother actions. Experiments show that bug fixes can reverse method rankings, moving the baseline from last to first on one task. Addressing design limitations can likewise remove anomalous gains: on another task, the baseline moves from 21 percentage points behind an accelerated method to 5 points ahead. Gains attributed to acceleration can therefore be artifacts of the benchmark rather than better task execution. We release our bug fixes and revised benchmark settings to support trustworthy evaluation of VLA acceleration.

1 INTRODUCTION

Vision-language-action (VLA) models (Kim et al., 2024; Shukor et al., 2025; Black et al., 2025; Physical Intelligence et al., 2025; NVIDIA, 2025) map visual observations and instructions to robot actions, and acceleration methods reduce their inference latency for on-robot deployment (Yang et al., 2025; Guan et al., 2025; Niu et al., 2026). These methods are judged by a simple protocol: report the speedup and show on standardized manipulation benchmarks (Chen et al., 2026b; Liu et al., 2023; Chen et al., 2026a; Zhou et al., 2025) that task success does not drop. The second half of this protocol rests entirely on the benchmark's success checker.

Training-free acceleration methods (Zhou et al., 2026; Fang et al., 2026) approximate the baseline policy's computation rather than adding capability, yet on some tasks they achieve higher success rates than the baseline itself. Such a gain means either that the approximation improves execution or that the checker scores the altered behavior more generously than the instruction warrants, and success rates alone cannot tell which. Therefore, we ask how benchmark flaws can make accelerated policies appear more successful than they are.

Answering this requires seeing what the robot did, not only the checker's verdict. Recording every rollout adds an 85% overhead on RoboTwin (Chen et al., 2026b), so we instead plot the object and end-effector trajectories recorded during evaluation against the checker's acceptance region and inspect videos only where this view flags a problem. This inspection reveals two patterns. In a *false baseline failure*, the baseline completes the instructed task but is rejected: in *Place Shoe*, the checker demands a toe orientation the prompt never mentions, failing the baseline while passing the

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

accelerated variant that reproduces the demonstrations' orientation (Figure 1). In a *false acceleration success*, the accelerated policy is accepted without completing the task: in *Put Bottles Dustbin*, it never grasps the bottle but knocks it off the table, the bottle rolls into the acceptance region, and the checker counts a success (Figure 2).

Tracing such cases across tasks and benchmarks, we find two kinds of flaws behind them. *Benchmark bugs* are places where the implementation departs from the intended task or evaluation protocol. They fall into three categories: instructions inconsistent with checkers and demonstrations, invalid initialization of prompts or scenes, and evaluation conditions that cannot be reproduced. Each corrupts the comparison differently, by scoring rollouts against unstated requirements, by changing the task before the policy acts, or by evaluating baseline and accelerated policies under different conditions.

Design limitations arise where the benchmark works as implemented but does not capture how acceleration changes execution. Checkers can be permissive: they may accept an object entering a target region without verifying the manipulation that put it there, and their distance thresholds make success rates sensitive to hyperparameters the instruction never exposes. Simulation can be idealized: unrealistic object masses let the same actions move an object differently in simulation and reality, and action smoothing that differs from real robots hides the jitter that acceleration adds to policy outputs.

Across seven benchmarks, this audit yields 22 bugs and 4 design limitations. We address them and re-evaluate the baseline against three accelerated variants. The corrections change conclusions, not only numbers. On LIBERO-PRO (Zhou et al., 2025), bug fixes raise the baseline's success rate on one task by 52.8 percentage points and move it from last to first; on RoboTwin, correcting object masses moves the baseline from 21 percentage points behind an accelerated variant to 5 points ahead.

Our contributions are threefold:

- We characterize misleading acceleration gains and introduce trajectory and acceptance-region visualization to provide evidence beyond success rates.
- We uncover 22 benchmark bugs and 4 design limitations.
- We re-evaluate acceleration after bug fixes and targeted design revisions, show that method rankings can reverse, and distill practical guidelines (Appendix D).

2 Related Work

Robotic Manipulation Benchmarks. Robotic manipulation benchmarks evaluate policies across diverse tasks and environments (James et al., 2019; Mees et al., 2021). LIBERO covers spatial, object, and goal variations as well as long-horizon tasks (Liu et al., 2023), while LIBERO-PRO and LIBERO-Plus extend its evaluation with task and environment perturbations to examine robustness (Zhou et al., 2025; Fei et al., 2026). Meta-World (Yu et al., 2019) and RoboTwin 2.0 (Chen et al., 2026b) cover multi-task and bimanual manipulation, while RoboCasa targets kitchen activities (Nasiriany et al., 2024) and VLABench emphasizes commonsense and long-horizon reasoning (Zhang et al., 2025). SIMPLER reduces visual and control gaps in simulation (Li et al., 2024), and RoboDojo unifies simulation and real-world evaluation (Chen et al., 2026a). These benchmarks commonly measure performance by success rates from binary task-success judgments, while RoboDojo additionally reports a graded score of partial task progress.

Training-free VLA Inference Acceleration. Training-free acceleration methods improve inference efficiency by mitigating computational redundancy. EfficientVLA combines language-model layer pruning, task-aware visual token selection, and intermediate-feature caching to reduce computation across multiple components of the VLA model (Yang et al., 2025). ProbeFlow and DP-Cache reduce inference latency by trimming redundant computations within the denoising phase of VLA models, whereas *Knowing When to Stop* lowers the overall inference frequency by adaptively tuning the number of actions executed per action chunk (Fang et al., 2026; Zhou et al., 2026; Xu et al., 2026). Typically evaluated against native baselines, these approaches use task success rate as a central measure of task performance. Such comparisons do not by themselves establish whether higher success rates reflect better execution of the task instructions.

Reliability of Robotic Manipulation Evaluation. RoboEval supplements binary success with measures of task progress, execution efficiency, bimanual coordination, and safety, revealing behavioral differences between policies with similar success rates (Wang et al., 2025). Jiang et al. (2026) audit five manipulation benchmarks and identify four threats to benchmark validity: shortcut solvability, insufficient evidence of statistical significance, creeping overfitting, and dependence on the source of training demonstrations. These issues weaken the connection between benchmark scores and manipulation capabilities. These works primarily examine evaluation reliability through behavioral measurement, task and test-distribution design, statistical analysis, and control of training data. We examine how benchmark corrections and evaluation refinements change comparisons between baseline and accelerated policies.

3 Characterizing the Evaluation of VLA Acceleration

This section introduces the components of VLA benchmarks and examines how instructions and demonstrations shape the behaviors assessed by success-rate metrics. We then use representative cases to illustrate their implications for evaluating VLA acceleration.

3.1 Benchmark Components

A simulated manipulation benchmark comprises simulation environment, task specifications, initialization procedures, success checkers, instruction construction, expert training demonstrations, and evaluation metrics. Table 1 summarizes instruction construction, training demonstration sources, and evaluation metrics in representative benchmarks. Across these simulation benchmarks, success is evaluated using rule-based checks on simulator states, with task-specific combinations of conditions, targets, and hyperparameter thresholds.

Table 1: Benchmark components. Instruction, Demos, and Evaluation denote instruction construction, training demonstration sources, and the success signal, respectively.

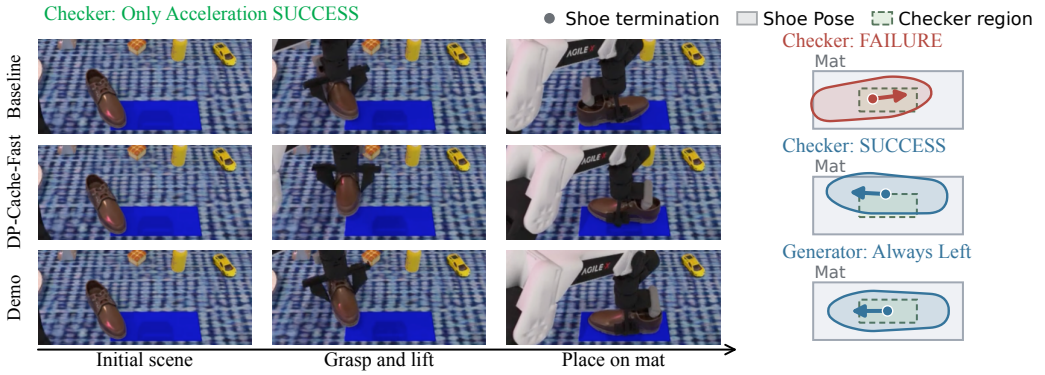
Benchmark	Instruction	Demos	Evaluation
LIBERO	Not specified	Human teleoperation	Binary
LIBERO-PRO	Based on LIBERO	Reused from LIBERO	Binary
LIBERO-Plus	LLM-rewritten	Reused from LIBERO	Binary
RoboTwin	LLM-generated	Scripted experts, motion planning	Binary
RoboCasa	Not specified	Human teleoperation, MimicGen	Binary
VLABench	LLM-rewritten	Scripted experts, motion planning	Binary, stage scores
RoboDojo	Not specified	Human teleoperation, motion planning	Binary, stage scores

The simulation environment governs physics and rendering, the task specification defines the intended behavior. Initialization procedures define the starting state, scenario, physical parameters, and control settings. Instructions and demonstrations influence the behavior being scored: the instruction specifies the requested task, and training demonstrations provide expert examples of its execution. Except for LIBERO variants, these benchmarks generate instructions by filling template slots with task-specific information. The evaluation metric records how often the resulting behavior satisfies the checker. The benchmark components provide the context for interpreting success-rate gains from acceleration.

3.2 How Benchmark Flaws Distort Acceleration Comparisons

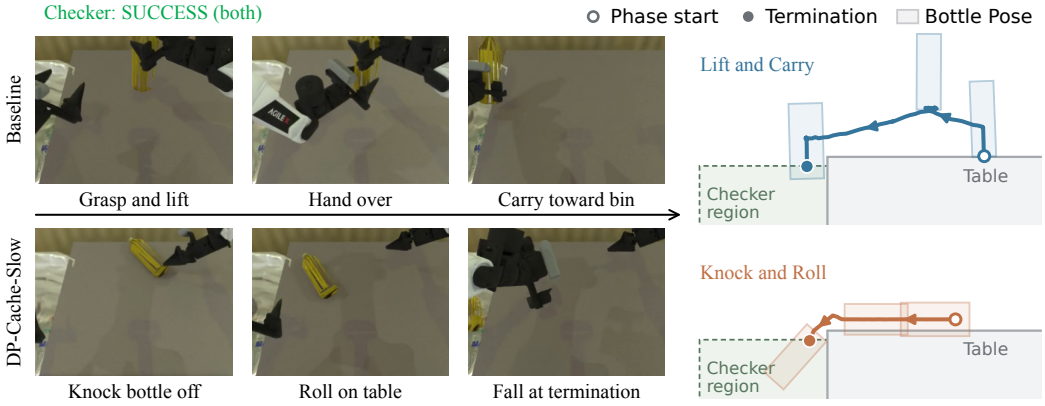
Training-free acceleration studies evaluate their methods on several manipulation benchmarks and report success-rate increases in some settings. ProbeFlow (Fang et al., 2026) evaluates on MetaWorld and LIBERO, with a small numerical increase on MetaWorld. DP-Cache (Zhou et al., 2026) evaluates on LIBERO and reports gains on its Spatial and Long suites. However, these studies do not report whether the additional rollouts counted as successful satisfy the task instructions beyond passing the benchmark success checkers.

Motivated by these reported gains, we examine whether benchmark flaws can change the measured performance of baseline and accelerated policies on RoboTwin. We inspect rollouts alongside the task instructions, available demonstrations, and checker outcomes. The following cases illustrate how checkers can reject valid task execution or accept behavior that violates the task instructions. Configurations of the acceleration methods are detailed in Section 7.1.



(a) Frames captured from the recording video (b) Visualization of Final Object Pose

Figure 1: Task *Place Shoe*. Simplified prompt: "Pick up the footwear from the table and place it on the mat." (a) Baseline and DP-Cache-Fast rollouts alongside a rule-based expert demonstration. (b) The final poses show that both shoes lie on the mat.



(a) Frames captured from the recording video (b) Visualization of trajectory

Figure 2: Task *Put Bottles Dustbin*. Simplified prompt: "Pick the bottle with two distinct parts, drop it into the garbage bin." (a) The recording rollout videos. (b) The trajectory visualizations show how both bottles enter the checker's acceptance region, serving as an alternative to rollout videos.

3.2.1 A Benchmark Bug Causes a False Baseline Failure

We evaluate acceleration methods on the *Place Shoe* task. In the example in Figure 1, starting from the same initial state, the baseline places the shoe facing right, while DP-Cache-Fast places it facing left. Both placements satisfy the prompt, but the checker rejects the baseline and accepts DP-Cache-Fast, enforcing a toe orientation that the prompt does not specify. DP-Cache-Fast's left-facing placement is learned from the rule-based training demonstrations. The checker follows the same orientation requirement as the demonstrations, causing it to reject the baseline's otherwise valid placement. After the benchmark fix, which also accepts right-facing placements, the baseline success rate rises from 65% to 74%, while DP-Cache-Fast rises from 68% to 69%. The baseline thus shifts from trailing DP-Cache-Fast by 3 percentage points to leading it by 5 points.

3.2.2 A Design Limitation Allows a False Acceleration Success

Figure 2 illustrates the *Put Bottles Dustbin* task with baseline and DP-Cache-Slow rollouts from the same initial scene. Although rolling the bottle off the table does not obey the instruction to pick it up, the rollout is still classified as successful. The checker tests whether the bottle enters the acceptance region, without verifying the instructed manipulation process. This case exposes a limitation of the success criterion: a higher reported score may not necessarily indicate better execution of the instructed manipulation. After adding a check that requires the gripper to be positioned above the bin within a time window before the bottle falls, the baseline success rate remains at 24%, whereas DP-Cache-Slow drops from 28% to 22%. The baseline thus shifts from trailing DP-Cache-Slow by 4 percentage points to leading it by 2 points.

These cases show how benchmark flaws produce anomalous gains by rejecting valid baseline behavior or accepting accelerated behavior that violates task instructions. Building on these observations, we investigate benchmark bugs and design limitations in Sections 5 and 6, respectively.

4 Workflow for Identifying Benchmark Flaws

To investigate the anomalous gains observed above, we use the following workflow to identify benchmark flaws.

Select tasks through anomalous gains. Success-rate gains motivate inspection but do not establish a flaw. First verify that baseline and accelerated runs use the same task instance and evaluation settings, investigating any initialization or reproducibility mismatch. For matched runs with different success signals, inspect both rollouts for false rejection or acceptance relative to the instruction.

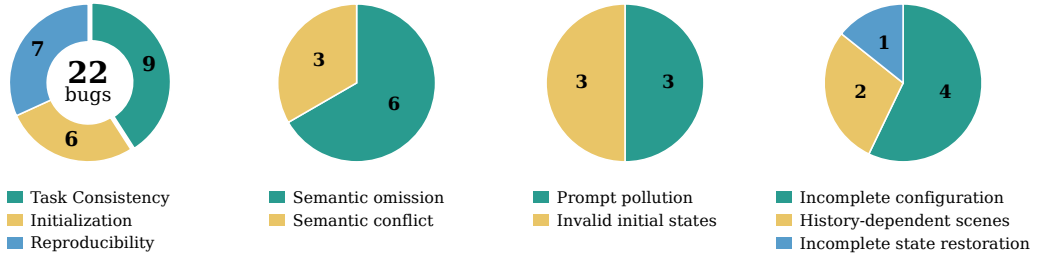
Localize the cause through lightweight visualization. Plot logged object and end-effector trajectories alongside gripper states and checker acceptance regions, marking success-trigger times. Compare these views with instructions and demonstrations, inspect the relevant code, and have domain experts manually verify suspected flaws. This view supports inspection without rendering and encoding every rollout, avoiding approximately 85% video-recording overhead (160.4 seconds/episode with recording vs. 86.4 without). Appendix A.5 reports detailed timing measurements.

Extend the audit by failure mechanism. Inspect tasks sharing the affected templates, loaders, checkers, or scene utilities, and components with equivalent roles in other benchmarks. Verify each case against its task specification and implementation, including tasks without anomalous gains.

By using acceleration methods to uncover benchmark flaws, we aim to enable fairer comparisons among variants of the same embodied intelligence model policy.

5 Localizing Benchmark Bugs

Following the workflow above, we identify 22 bugs across the 7 benchmarks in Table 1. These bugs fall into 3 categories: task consistency, initialization, and reproducibility. Section 7.2 compares evaluation results before and after bug fixes.



(a) All categories (b) Task Consistency (c) Initialization (d) Reproducibility

Figure 3: Distribution of the 22 benchmark bugs. (a) Counts across all categories; (b)–(d) subcategory counts within task consistency, initialization, and reproducibility, respectively.

5.1 Task Consistency: Instruction–Checker–Demo Misalignment

As Table 1 shows, some benchmarks use LLM to expand task instructions, increasing linguistic diversity for evaluating generalization. However, diverse wording does not guarantee preservation of task semantics, and combining language templates with scenario information can introduce unexpected conflicts. Figure 3(b) summarizes these inconsistencies.

(1) Semantic omission. Some instructions omit constraints retained in the demonstrations and success checkers. As shown earlier, *Place Shoe*'s checker restricts toe orientation beyond the instruction's placement requirement. Relaxing this orientation constraint aligns the checker with the semantics of the delivered prompt. In RoboTwin's *Handover Block* task, the prompt is "Grab the red block with the left arm." It omits the handover and placement behavior shown in the demonstrations: the left arm passes the block to the right arm, which places and releases it on the pad. The checker requires placement on the pad and an open right gripper, rejecting grasp-only completion. Similar omissions occur in RoboTwin's *Blocks Ranking Size*, *Rotate QRCode* and *Adjust Bottle*, and VLABench's *Select Mahjong*.

(2) Semantic conflict. Instructions, checkers, and demonstrations specify different operations. In RoboTwin's *Place Bread Basket* task, some instructions require two-arm grasping of adjacent bread, but the corresponding demonstrations use one arm. The checker checks final bread positions without verifying arm use, allowing one-arm execution to pass. In RoboTwin's *Place Bread Skillet*, some instructions request left-arm bread transport, but the corresponding demonstrations use the right arm, and the checker does not verify which arm is used. In VLABench's *Select Painting*, the instruction asks the robot to grasp a painting, but the demonstration presses a button and passes the checker.

These cases show that instructions, demonstrations, and checkers do not always specify the same task requirements. As a result, the success signal may not accurately reflect whether a policy follows the given instruction.

5.2 Initialization: Prompt and Initial-State Errors

Section 5.1 concerns instructions that misstate or omit task requirements. Here, the error occurs during initialization: the loader selects the wrong task description or adds unrelated text, or pairs a scene with incompatible object states, as shown in Figure 3(c).

(1) Prompt pollution. The received instruction does not match the task. In LIBERO-PRO's *Turn On The Stove* task, the policy receives the "turn on the stove" prompt, but the checker requires it off. The loader derives the prompt from an unchanged filename instead of the updated task definition. LIBERO-Plus appends task-specific metadata as instruction suffixes or mistakes prompts written for LLM-based rewriting for task instructions.

(2) Invalid initial states. The initial state violates intended scene conditions. In a LIBERO-PRO *Environment Perturbation* task, mismatched generation and loading directories cause a new scene to load an old state file without validating object positions. When a living-room tabletop scene is replaced with a kitchen scene, objects retain their original positions, fall from midair, and bounce off the table, preventing task completion. In RoboCasa, source and destination counters can resolve to the same physical object, making transport successful at initialization. RoboDojo's *Organize Table* requests placing remaining objects in a drawer, but no such objects exist on the tabletop.

These defects alter the task presented to the policy or its initial conditions, potentially making a task impossible to complete or already successful before execution. The resulting scores can therefore distort comparisons between baseline and accelerated policies on the intended tasks.

5.3 Reproducibility: Uncontrolled Randomness and Incomplete Replay

To isolate the effects of acceleration, comparisons must hold non-policy variables fixed. We identify 3 types of implementation defects that compromise this control, as summarized in Figure 3(d).

(1) Incomplete evaluation configuration. Identical seeds can yield different evaluation conditions. In RoboTwin, the same scene can receive different instructions because the episode seed does not control instruction sampling. VLABench similarly fails to pass the requested seed to its independent scene generator. LIBERO-Plus samples non-default task orders before setting the seed, allowing identical configurations to produce different task sequences. LIBERO leaves MuJoCo unpinned, allowing version changes to alter rendered observations.

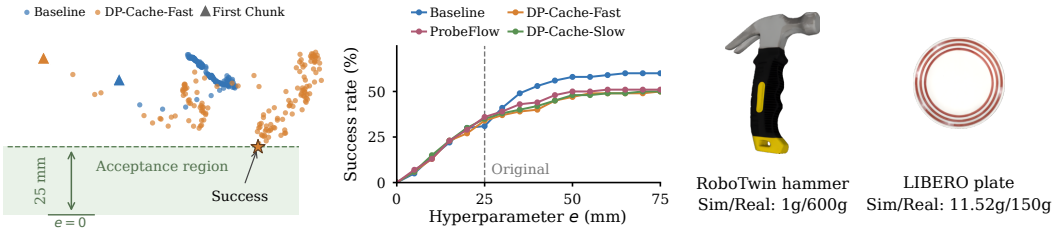
(2) History-dependent scene construction. Matching trial indices need not identify the same scene. In RoboCasa, the environment generates a scene for each trial. Even when two runs start with the same seed, they can generate different scenes for the same trial. In RoboTwin, the same seed can produce different clutter positions depending on how many episodes have already run in the process. Matching trial indices thus does not ensure matched scenes.

(3) Incomplete state restoration. In some RoboCasa tasks, reloading a saved successful state can produce a failure judgment. The robot and objects return to their saved positions, but the checker still uses target positions from a different scene. Restoring the visible scene therefore does not restore the conditions used to judge success.

These problems break consistency in inputs, scenes, or scoring conditions. Identical seeds, trial indices, or even saved states are therefore insufficient to ensure matched evaluation conditions. Measured success-rate differences can include effects of non-policy variables, making them difficult to attribute specifically to acceleration methods.

6 BENCHMARK DESIGN LIMITATIONS

Beyond implementation bugs, we examine two categories of benchmark design limitations: permissive success checkers and simulation-to-reality gaps. Section 7.3 evaluates how addressing these limitations affects policy comparisons.



(a) Spatial distributions (b) Threshold sensitivity (c) Object masses

Figure 4: Benchmark design limitations arising from fixed acceptance thresholds and unrealistic object masses. (a,b) Spatial distributions of *Scan Object* samples relative to the acceptance region and success rates under different acceptance thresholds, respectively. (c) Discrepancies between simulated and real-world object masses.

6.1 PERMISSIVE CHECKERS

(1) Incomplete checker criteria. Current checkers incompletely capture the execution process, allowing success without verifying how the goal was reached, as detailed in Figure 2. In VLABench's *Select Drink* task, the checker only requires the target drink to leave the container, so a juice carton dropped during extraction still counts as a success once it leaves the fridge region.

(2) Reliance on fixed acceptance thresholds. A single acceptance threshold may bias policy comparisons. In *Scan Object*, distance thresholds define the acceptance region used to determine success. As shown in Figure 4(a), which plots the spatial distributions from one rollout per policy, DP-Cache-Fast exhibits variances in axial distance and transverse error e that are $11.8\times$ and $4.5\times$ those of the baseline, respectively. Given different spatial distributions, success rates may depend on acceptance thresholds. To assess whether the policy comparison is robust to threshold choice, we sample the parameters defining the acceptance region and re-evaluate across these settings (Figure 4(b)).

6.2 SIMULATION-TO-REALITY GAPS IN PHYSICAL EXECUTION

Beyond visual variation addressed by domain randomization (Tobin et al., 2017), simulation fidelity depends on action controllers and object properties.

(1) Unrealistic object masses. In RoboTwin, over 60% of the audited dynamic object creation sites assign the default mass of 10 g, while even individually specified masses can be unrealistic: the hammer in *Beat Block Hammer* is assigned only 1 g, compared with its real-world mass of 600 g. RoboCasa and LIBERO compute masses for most objects from collision geometry using a density of 100 kg/m^3, which may not reflect their actual materials. For example, a plate in LIBERO is assigned a mass of just 11.52 g, compared with its real-world mass of 150 g. These discrepancies illustrate how visually realistic objects can have implausible physical properties, as shown in Figure 4(c). Such mass settings can alter object responses to the same policy actions and contribute to the simulation-to-reality gap (Peng et al., 2018); Table 3 reports the evaluation results.

(2) Differences in action processing. We compare policy outputs and executed motion on *Pick Dual Bottles* in simulation and on a real robot (Figure 5). Following prior smoothness analyses (Zhao et al., 2025; Wang et al., 2025), we measure executed acceleration (rad/s^2) and executed jerk magnitude (rad/s^3). For denormalized outputs, action acceleration (rad/index^2) and action jerk magnitude (rad/index^3) are computed over action indices rather than time, before post-processing.

In simulation, DP-Cache-Slow and ProbeFlow have mean action jerk magnitudes of $1.45\times$ and $2.16\times$ the baseline's, respectively, while their mean executed jerk magnitudes are only $0.94\times$ and $0.92\times$ the corresponding baseline value. On the real robot, the latter ratios rise to $1.35\times$ and $1.43\times$. Thus, the smoothness advantage in simulation does not necessarily transfer to real execution.

In simulation, time-optimal path parameterization (TOPP) (Pham & Pham, 2017) retimes paths under motion constraints, changing execution duration and potentially the measured jerk. Control-

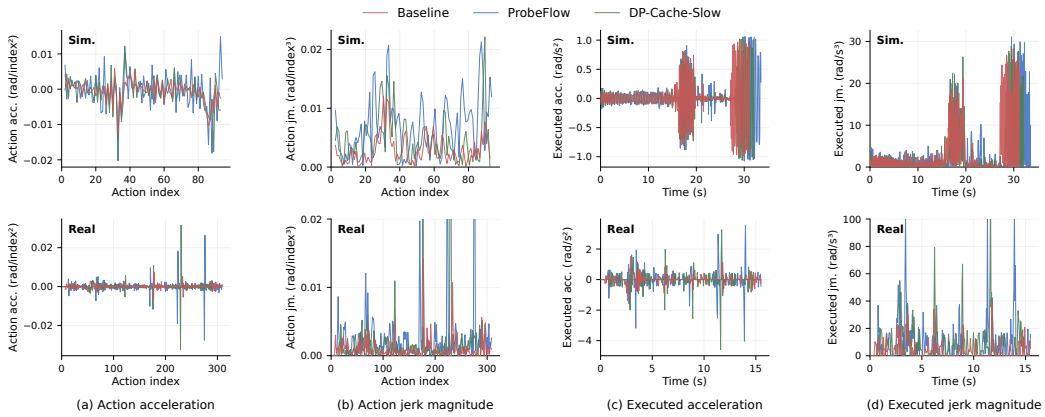


Figure 5: Policy-action fluctuations and executed motion in simulation (top) and on the real robot (bottom). Acceleration and jerk magnitude are abbreviated as acc. and jm., respectively.

frequency limits and differences in low-level motor-control implementations may explain the discrepancy between simulation and real execution. We therefore use action smoothness before post-processing as a complementary ranking criterion for on-robot deployment. For each paired episode, we rank all four methods by action variation in ascending order and assign rank-based scores, with failed rollouts receiving zero points. Appendix B provides the details. This score favors smoother policy outputs by accounting for action fluctuations before post-processing.

7 A Trustworthy Evaluation of VLA Acceleration

Our evaluation addresses two questions:

$\mathcal{Q}\mathbf{1}$: How do benchmark bug fixes affect the measured gains of VLA acceleration methods? (§ 7.2)
$\mathcal{Q}\mathbf{2}$: How do checker design and physical parameters shape the interpretation of these gains? (§ 7.3)

7.1 Experimental Setup

Hardware and models. Our evaluation infrastructure uses NVIDIA A800 and RTX 4090 GPUs. We build on public benchmark implementations and community-released $\pi_{0.5}$ (Physical Intelligence et al., 2025) checkpoints. Further hardware and software details are provided in Appendix A.

Benchmarks and Acceleration methods. We audit seven benchmarks (Section 5) and evaluate the tasks listed below. We compare Baseline with DP-Cache-Fast ($S = 5$), DP-Cache-Slow ($S = 2$) (Zhou et al., 2026), and ProbeFlow ($\epsilon = 0.008$) (Fang et al., 2026), which achieve speedups of $2.56\times$, $1.62\times$, and $1.49\times$, respectively. Configuration details are shown in Appendix A.2.

Metrics. We report success rates (SR) and gains over the baseline before and after revision. For motion-aware scoring, revised values report graded scores rather than success rates. Measurement details are provided in Appendix A.4. Additional results are shown in Appendix C.

7.2 Bug Fixing Results

Table 2 shows that bug fixes can reverse method rankings. On RoboTwin's *Rotate QRCode*, the baseline shifts from 3 percentage points behind ProbeFlow to 6 ahead. On LIBERO-PRO's *Spatial-Task*, baseline SR rises from 0.4% to 53.2%, moving from last to first.

Extending the audit to VLABench shows effects even when the baseline remains first. On *Select Mahjong*, baseline SR gains 17 percentage points, versus 6–10 for the accelerated methods. On *Select Painting*, the baseline gains 13 points. Thus, bug fixes can also widen the baseline's lead without changing the top-ranked method.

7.3 Results of Addressing Benchmark Limitations

7.3.1 Permissive Checkers

The *Put Bottles Dustbin* case is detailed in Section 3.2.2. On VLABench's *Select Drink* (Table 3), the original checker gives the baseline an SR of 35% and ProbeFlow 19%. After requiring controlled removal, baseline SR drops to 31%, whereas ProbeFlow drops to 9%. The baseline's lead thus widens from 16 to 22 percentage points.

Table 2: Original / revised SR (%); blue/pink marks each row's worst/best values. The listed LIBERO-PRO and LIBERO-Plus tasks are enhanced variants of a subset of LIBERO tasks.

Benchmark	Task / subset	Baseline	DP-Cache-Fast	DP-Cache-Slow	ProbeFlow
RoboTwin	*place shoe*	65 / 74	68 / 69	66 / 67	68 / 69
	rotate qrcode	56 / 66	60 / 62	48 / 57	59 / 60
	...				
	blocks ranking size	21 / 27	22 / 25	16 / 21	19 / 23
	handover block	18 / 29	15 / 26	20 / 27	16 / 25
VLABench	*select mahjong*	18 / 35	7 / 17	14 / 20	8 / 17
	select painting	9 / 22	8 / 21	8 / 19	6 / 14
LIBERO-PRO	*Object-Position*	16.8 / 18.6	17.0 / 17.6	15.8 / 17.0	17.4 / 17.2
	Spatial-Position	46.4 / 51.4	45.6 / 50.6	46.0 / 51.0	47.8 / 50.6
	...				
	Object-Task	0.0 / 10.6	0.0 / 10.6	0.0 / 10.0	0.0 / 10.6
	Spatial-Task	0.4 / 53.2	0.4 / 52.8	0.6 / 52.2	0.6 / 52.2

For *Scan Object* in Figure 4(b), we vary the transverse tolerance e, keeping other checker conditions fixed. Below 25 mm, methods achieve similar success rates; above 25 mm, the baseline outperforms all accelerated methods, especially at larger thresholds. At 25 mm, the baseline trails ProbeFlow by 5 percentage points (31% vs. 36%); at 50 mm, it leads by 8 points (58% vs. 50%).

7.3.2 Mass Sensitivity and Motion-Aware Scoring

Table 3 presents selected tasks for mass adjustment and motion-aware scoring. With the checkpoint and scoring rule held fixed, mass adjustment reduces baseline SR on *Place Fan* from 40% to 38%, while DP-Cache-Fast drops from 61% to 33% (Table 3). The baseline thus moves from 21 percentage points behind DP-Cache-Fast to 5 points ahead. Training with the corrected mass further raises baseline SR from 38% to 53%, outperforming DP-Cache-Fast (33%-45%) by 8 percentage points.

After motion-aware scoring, the baseline moves from 5 percentage points behind ProbeFlow to 16.6 points ahead on *Scan Object*. On *Turn Switch*, baseline SR remains at 25%, while DP-Cache-Slow drops from 31% to 24.8%. These reversals highlight the importance of realistic physical parameters and action-processing settings for reliable policy rankings.

Table 3: Original SR (%) / revised SR or scores; blue/pink marks each row's worst/best values.

Benchmark	Task	Baseline	DP-Cache-Fast	DP-Cache-Slow	ProbeFlow
Permissive checkers					
RoboTwin	*put bottles dustbin*	24 / 24	26 / 22	28 / 22	28 / 22
VLABench	*select drink*	35 / 31	17 / 12	20 / 17	19 / 9
Mass adjustment					
RoboTwin	*beat block hammer*	79 / 27	87 / 19	82 / 19	86 / 22
	place fan	40 / 38	61 / 33	51 / 38	59 / 34
LIBERO	*Mug Placement in Microwave*	94 / 60	98 / 56	94 / 52	98 / 50
	Two-Mug Placement	98 / 88	94 / 82	94 / 82	94 / 80
Motion-aware scoring					
RoboTwin	*scan object*	31 / 31.0	34 / 20.0	35 / 28.0	36 / 14.4
	turn switch	25 / 25.0	25 / 12.8	31 / 24.8	27 / 13.2

8 Conclusion

This work systematically investigates benchmark flaws to enable trustworthy comparisons of VLA acceleration methods. Through a comprehensive evaluation across seven benchmarks, we identify 22 bugs and 4 limitations involving instruction–checker mismatches, initialization defects, incomplete reproducibility, success checks, physical parameters, and action-processing mechanisms. We report these findings and contribute fixes to the community. We will continue investigating flaws in emerging benchmarks and tracking the progress of submitted pull requests to support more reliable evaluation of VLA acceleration methods.

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and language refinement, and to help identify and retrieve relevant literature. We reviewed the AI-assisted text and verified that the cited sources support the corresponding claims. All reported experimental results came from benchmark evaluations rather than AI-generated data. We take full responsibility for the final content of this work, including its text, claims, and supporting materials.

ETHICS STATEMENT

Our work examines the reliability of robotic manipulation benchmarks and the evaluation of policy acceleration methods. We use publicly available benchmarks and model checkpoints, together with task demonstrations collected for our experiments. We document benchmark defects and evaluation limitations to support reproducible comparisons and reduce misleading conclusions about policy performance. Our findings concern the evaluated tasks and conditions; success in simulation does not establish safety or reliability in real-world deployment. We encourage appropriate safety validation and human oversight when deploying accelerated policies on physical robots.

REPRODUCIBILITY STATEMENT

This study audits published benchmarks and evaluates existing acceleration methods. We describe the evaluation setup in Section 7.1, with hardware and model details in Appendix A, method configurations in Appendix A.2, and measurement details in Appendix A.4. Appendix A.1 lists the repository commits used for the audited benchmarks. Our implementations of the acceleration methods follow publicly released code and the original papers. Appendix B documents the identified bugs and their repairs, along with changes to checker criteria and physical parameters. The supplementary material includes bug fixes and changes addressing design limitations. After the anonymous review period, we will add links in the Appendix to our submitted issues and pull requests.

REFERENCES

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A Unified Latent Action World Model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 35101–35113, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Bi_Motus_A_Unified_Latent_Action_World_Model_CVPR_2026_paper.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Robotics: Science and Systems (RSS)*, 2025. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Weijie Wan, Baijun Chen, Haoran Lu, Haowen Yan, Honghao Su, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026a. URL https://arxiv.org/abs/2607.04434.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2026b. URL https://icml.cc/virtual/2026/poster/62192.

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

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. LIBERO-Plus: A Progressive Robustness Benchmark for Visual-Language-Action Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 38574–38583, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Fei_LIBERO-Plus_A_Progressive_Robustness_Benchmark_for_Visual-Language-Action_Models_CVPR_2026_paper.html.

Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile ALOHA: Learning Bimanual Mobile Manipulation Using Low-Cost Whole-Body Teleoperation. In *Conference on Robot Learning (CoRL)*, pp. 4066–4083, 2024. URL https://proceedings.mlr.press/v270/fu25b.html.

Weifan Guan, Qinghao Hu, Aosheng Li, and Jian Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey, 2025. URL https://arxiv.org/abs/2510.17111.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *arXiv preprint arXiv:1909.12271*, 2019. URL https://arxiv.org/abs/1909.12271.

Tianchong Jiang, Xiangshan Tan, Samuel Wheeler, Luzhe Sun, Tewodros W. Ayalew, and Matthew Walter. What are we actually benchmarking in robot manipulation? *arXiv preprint arXiv:2606.04233*, 2026. URL https://arxiv.org/abs/2606.04233.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. In *Conference on Robot Learning (CoRL)*, pp. 2679–2713, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024. URL https://arxiv.org/abs/2405.05941.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks. *arXiv preprint arXiv:2112.03227*, 2021. URL https://arxiv.org/abs/2112.03227.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-Scale Simulation of Household Tasks for Generalist Robots. In *Robotics: Science and Systems (RSS)*, 2024. doi: 10.15607/RSS.2024.XX.050. URL https://www.roboticsproceedings.org/rss20/p050.html.

Jiahui Niu, Kefan Gu, Yucheng Zhao, Shengwen Liang, Tiancai Wang, Xing Hu, Ying Wang, and Huawei Li. Realtime-vla flash: Speculative inference framework for diffusion-based vlas. *arXiv preprint arXiv:2605.13778*, 2026.

NVIDIA. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, 2018. doi: 10.1109/ICRA.2018.8460528. URL https://doi.org/10.1109/ICRA.2018.8460528.

Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis, 2017. URL https://arxiv.org/abs/1707.07239.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. In *Conference on Robot Learning (CoRL)*, pp. 17–40, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. *arXiv preprint arXiv:2506.01844*, 2025. URL https://arxiv.org/abs/2506.01844.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. doi: 10.1109/IROS.2017.8202133. URL https://doi.org/10.1109/IROS.2017.8202133.

Yi Ru Wang, Carter Ung, Christopher Tan, Grant Tannert, Jiafei Duan, Josephine Li, Anh Le, Rishabh Oswal, Markus Grotz, Wilbert Pumacay, Yuquan Deng, Ranjay Krishna, Dieter Fox, and Siddhartha Srinivasa. RoboEval: Where robotic manipulation meets structured and scalable evaluation. *arXiv preprint arXiv:2507.00435*, 2025. URL https://arxiv.org/abs/2507.00435.

Runze Xu, Xiaolong Shan, Shuang Dai, Yu Wang, and Jincheng Yu. Knowing when to stop: Adaptive action chunking via internal cross-attention dynamics in vlas. *arXiv preprint arXiv:2609.00908*, 2026. URL https://arxiv.org/abs/2609.00908.

Yantai Yang, Yuhao Wang, Zichen Wen, Zhongwei Luo, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-free acceleration and compression for vision-language-action models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. doi: 10.52202/085713-1365. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3a2ef31a1e45908901adc0ca853a8faf-Abstract-Conference.html.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, pp. 1094–1100, 2019. URL https://proceedings.mlr.press/v100/yu20a.html.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. VLABench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Zhang_VLABench_A_Large-Scale_Benchmark_for_Language-Conditioned_Robotics_Manipulation_with_Long-Horizon_ICCV_2025_paper.html.

Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots, 2025. URL https://arxiv.org/abs/2512.24673.

Kaijun Zhou, Qiwei Chen, Da Peng, Zhiyang Li, Xijun Li, and Jinyu Gu. Characterizing vision-language-action models across XPUs: Constraints and acceleration for on-robot deployment. In *International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65223.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. 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.

A HARDWARE AND MODEL SETUP

Our evaluation infrastructure uses NVIDIA A800 and RTX 4090 GPUs. We build on public benchmark implementations and community-released $\pi_{0.5}$ checkpoints adapted to the corresponding tasks. Each method comparison uses the same checkpoint and observation/action interface within a benchmark. For RoboTwin, we use the $\pi_{0.5}$ checkpoint released by the Motus team (Bi et al., 2026).[1]

Our real-robot experiments use the Mobile ALOHA platform (Fu et al., 2024).

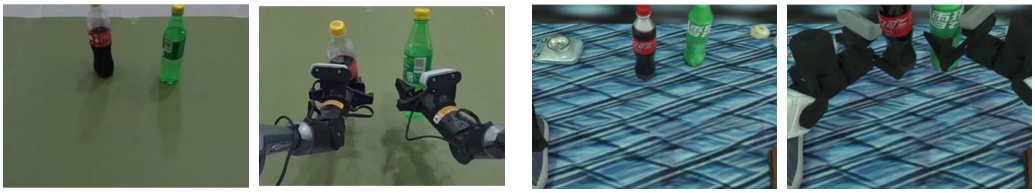


Figure 6: *Pick Dual Bottles* on the real robot (left pair) and in simulation (right pair).

A.1 BENCHMARK AND EVALUATION FRAMEWORK VERSIONS

Table 4 records the repository revisions used in our evaluations. LeRobot and OpenPI are listed separately as evaluation and policy frameworks.

Table 4: Repository revisions used for benchmark evaluation.

Repository	Commit SHA
RoboTwin	`2eeec322d95799f537cbfe5f291a8220d965ccb8`
LIBERO	`8f1084e3132a39270c3a13ebe37270a43ece2a01`
LIBERO-PRO	`eafdb809426b13153aa1e4c42d6601844217dfec`
RoboCasa365	`be22d659b02db8f6d7f3a3c3edc742934fdcbaae`
LIBERO-Plus	`4976dc30028e805ff8094b55501d532c48fec182`
VLABench	`cf588fe60c0c7282174fe979f5913170cfe69017`
RoboDojo	`ee67a1468510da7624a089164402359f2afc72c8`

A.2 METHOD CONFIGURATIONS AND COMPARISON PROTOCOL

DP-Cache. Following the notation of Zhou et al. (2026), S denotes the cache interval: one recomputation per S steps, with cached outputs reused on the remaining steps. DP-Cache-Fast and DP-Cache-Slow use $S = 5$ and $S = 2$, respectively.

ProbeFlow. We use linearity tolerance $\epsilon = 0.008$, integration-step bounds $N_{\min} = 2$ and $N_{\max} = 10$, step increment $\Delta N = 2$, and lookahead probe horizon $\Delta t_{\text{probe}} = 0.5$, following the settings reported by Fang et al. (2026).

Comparison protocol. Repairs are compared within the same task cohort. When only the checker changes, the comparison can use two scores from the same recorded trajectory; prompt, scene, or physical parameter changes require a new rollout under the corresponding condition.

Training details. For the 3,000-step *place fan* fine-tuning run, we collect 100 successful expert demonstrations with a 600 g fan and a 20 g target pad. We initialize from the existing Motus $\pi_{0.5}$ checkpoint. Training uses one NVIDIA A800, batch size 4, a peak learning rate of 2.5×10^{-5}, and 100 warm-up steps.

[1] `https://huggingface.co/motus-robotics/pi0.5_robotwin2`

A.3 Inference Latency

We measure OpenPI PyTorch inference on one NVIDIA RTX 4090 with compilation disabled, using the original RoboTwin $\pi_{0.5}$ checkpoint. Each configuration receives 20 warm-up requests followed by 200 timed requests, with batch size 1 and action horizon 32. Inputs are fixed synthetic three-camera observations (224×224), state, and a task prompt.

Table 5: Mean inference latency and speedup.

Configuration	Latency (ms)	Speedup
Baseline	313.73	1.00×
DP-Cache-Fast	122.61	2.56×
DP-Cache-Slow	194.12	1.62×
ProbeFlow	210.87	1.49×

A.4 Measurement Details

Output-action variation and executed joint jerk are reported separately, using action indices and physical timestamps, respectively. Mass comparisons use original and adjusted parameter settings while recording whether inertia is also changed. A dash denotes an unfilled result, not zero success or a completed experiment.

A.5 Video-Recording Overhead

On device-local frozen *place fan* cases, enabling additional video recording increases mean episode latency from 86.42 s to 160.26 s on an A800 (85.4%) and from 24.22 s to 39.96 s on an RTX 4090 (65.0%). Policy-input rendering remains enabled in both conditions.

B Benchmark Bug Repairs and Design Limitations

Table 6 lists the 22 bugs identified in Section 5 and their corresponding repairs, grouped by category and subcategory. Figure 7 illustrates a process-aware checker and the effect of physical-parameter changes.

Table 6: Benchmark bugs and corresponding repairs.

No.	Benchmark	Bug	Repair
Task consistency — Semantic omission			
1	RoboTwin	*place shoe*: the checker and demonstrations restrict toe orientation, but the instruction only specifies mat placement.	Remove the unrequested orientation constraint; retain the mat-placement check.
2	RoboTwin	*rotate qrcode*: the checker adds table-height and gripper-release requirements absent from the instruction.	Remove the height and release constraints; retain the orientation check.
3	RoboTwin	*adjust bottle*: the checker restricts lateral position by initial side, although the prompt only requests grasping and lifting.	Accept either lateral region when no arm is specified; see Figure 8 for arm-selection analysis.
4	RoboTwin	*blocks ranking size*: the instruction omits parts of the ordering rule used by the checker and demonstrations.	Explicitly state the required size order and placement in the instruction.
5	RoboTwin	*handover block*: grasp-only prompts are scored against a longer handover, placement, and release sequence.	Adapt the checker to the delivered prompt and check only its required operations.
6	VLABench	*select mahjong*: a pickup instruction is scored against an additional mat placement requirement.	remove the unrequested mat-placement condition.

Continued on next page

Table 6 (continued)

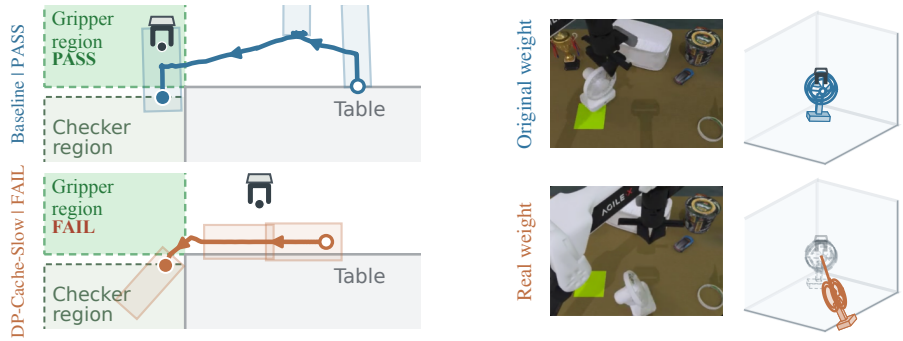
No.	Benchmark	Bug	Repair
Task consistency — Semantic conflict			
7	RoboTwin	*place bread skillet*: prompts specify the left arm, demonstrations use the right, and arm use is not checked.	Align the specified arm with demonstrations and add an arm-use check.
8	RoboTwin	*place bread basket*: dual-arm prompts conflict with single-arm demonstrations; only final positions are checked.	Check arm use against the prompt's single- or dual-arm requirement.
9	VLABench	*select painting*: prompts request grasping a painting, but demonstrations and scoring use a button press.	For grasping prompts, check a grasp of the target painting instead of a button press.
Initialization — Prompt pollution			
10	LIBERO-PRO	Prompts come from stale filenames rather than updated BDDL instructions, causing conflicts such as stove on versus off.	Read the instruction from the current task's BDDL language field. Evaluation results are reported in Tables 2 and 7.
11	LIBERO-Plus	BDDL language fields contain LLM-rewriting preambles.	Correct the task language in the affected BDDL files.
12	LIBERO-Plus	Filename-derived prompts include camera views, initial-state indices, or other configuration metadata.	Read task language from BDDL to exclude filename metadata from policy inputs. Evaluation results are reported in Table 7.
Initialization — Invalid initial states			
13	LIBERO-PRO	Perturbed scenes load incompatible old states, leaving object positions inconsistent with the new scene.	Align generation/loading directories, load matching states, and check initial spatial relations.
14	RoboCasa	*ArrangeDrinkware*: identical source and destination counters make the task successful at initialization.	Resample initially successful instances. After repair, $\pi_{0.5}$ SR drops from 20% to 0%.
15	RoboDojo	*organize table*: the prompt requests drawer placement of objects absent from the initial scene.	Add the objects required for drawer placement and the corresponding success checks.
Reproducibility — Incomplete evaluation configuration			
16	RoboTwin	Instruction sampling is not fully seeded, so identical episode seeds can yield different prompts.	Bind instruction-generation and selection randomness to the episode seed.
17	VLABench	The environment loader omits the random state required by the independent scene generator.	Pass a seed-controlled random state to the scene generator.
18	LIBERO-Plus	Non-default task orders are sampled before setting the command-line seed.	Set the seed before generating the task order.
19	LIBERO	Unpinned MuJoCo versions produce different rendered policy inputs.	Pin MuJoCo to 3.3.2 instead of 3.8.1; $\pi_{0.5}$ SR on *pick up bowl* rises from 62% to 88%.
Reproducibility — History-dependent scene construction			
20	RoboTwin	In-place changes to clutter bounds accumulate across episodes, making scenes depend on initialization history.	Initialize from a fresh copy of the original bounds to prevent accumulated offsets.
21	RoboCasa	History-dependent resets can produce different scenes for the same seed and trial index.	Add a reset interface with explicit per-episode seeding.
Reproducibility — Incomplete state restoration			
22	RoboCasa	Restored scenes can be scored against targets from a newly sampled scene, reversing a saved success label.	Save and restore checker targets and other task state alongside the simulation state.

Process-aware dustbin checker. For *put bottles dustbin*, we measured the time for a bottle to fall into the bin from positions near the table edge, at heights ranging from tabletop level to 30 cm above it, and obtained a 250 ms interval. We therefore augment the original bottle-position checks with a per-bottle process check: during the 250 ms preceding entry into the checker region, the gripper

held the bottle must have appeared in the space directly above that region. Applying this condition to each bottle adds evidence of delivery by the robot rather than accepting position alone. Figure 7(a) illustrates one bottle: Baseline satisfies the additional check, whereas DP-Cache-Slow does not.

Acceptance-threshold sensitivity. In *scan object*, d limits axial separation and e bounds transverse alignment error component-wise. We fix $d = 70$ mm (default) and, among states satisfying all other checks, record each episode's minimum required tolerance $e_{\min}$. Figure 4(b) plots the fraction with $e_{\min} < e$ as e varies.

Mass-sensitive grasping. Figure 7(b) illustrates the effect of correcting the fan mass in *place fan*. A grasp near the fan's edge can succeed in simulation with the original 10 g mass. After the fan mass is corrected to 600 g, the illustrated grasp no longer retains the fan: it slips and falls, and the task fails.



(a) Additional process check

(b) Physical-parameter effect

Figure 7: (a) The gripper-presence check accepts Baseline and rejects DP-Cache-Slow. (b) An edge grasp holds the 10 g fan but fails at 600 g. Gray dashed poses and grippers are schematic references; pad mass and inertia also change.

Motion-aware ranking. For each paired episode, we rank all four methods by the mean absolute action jerk magnitude of their policy-output actions over time and the 12 arm joints, excluding grippers. Ranks 1–4 receive 1.0, 0.8, 0.6, and 0.4 points (ties share the average); failures participate in ranking but receive zero points. We report 100 times the mean episode score, alongside SR, in Tables 3 and 8.

Task consistency: a closer look at arm selection during denoising. We further investigate the task-consistency case study through a more detailed analysis of the denoising process. Figure 8 examines the first action chunk of an *adjust bottle* case across ten flow-matching denoising steps. TCP denotes the *tool center point*, the reference point at the gripper end effector. From Step 2 onward, the dominant predicted motion consistently shifts to the left arm, while the right arm remains nearly stationary. As denoising proceeds, the predicted left-arm path becomes visually smoother overall. This case illustrates how denoising can change arm selection and refine motion, motivating the prompt-aligned lateral acceptance rule described in entry 3 of Table 6.

C Additional Success-Rate Comparisons and Visualizations

Table 7 supplements the bug-fix comparisons in Table 2. Figure 9 provides visual examples of instruction–checker mismatches and permissive success checks. Table 8 adds mass-adjustment and motion-aware scoring results omitted from Table 3. Entries report before / after SR (%), except motion-aware rows, which report SR (%) / score (0–100).

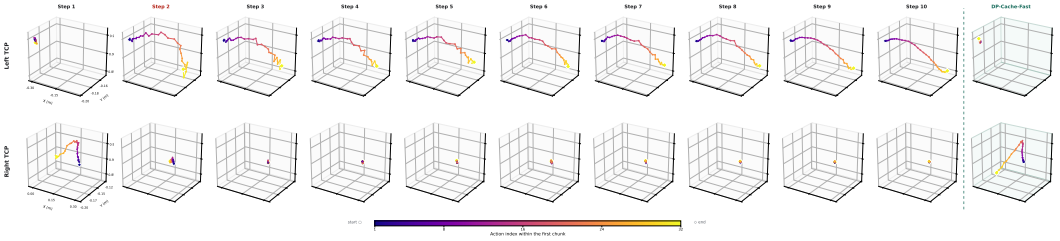


Figure 8: Step-by-step flow-matching denoising on *adjust bottle*. Top/bottom rows show left/right TCP paths for the first 32-action chunk; color indicates action order within the chunk. The rightmost column shows the accelerated comparison. From Step 2, motion remains concentrated in the left arm and becomes smoother over subsequent denoising steps.

Table 7: Additional bug-fix comparisons: original / repaired SR (%); blue/pink marks each row's worst/best values separately before and after repair.

Benchmark	Task	Baseline	DP-Cache-Fast	DP-Cache-Slow	ProbeFlow
Task consistency					
RoboTwin	*adjust bottle*	92 / 96	94 / 94	95 / 95	94 / 94
	place bread skillet	42 / 35	47 / 32	39 / 29	46 / 32
	place bread basket	55 / 54	59 / 53	55 / 52	58 / 52
Initialization					
LIBERO-PRO	*Goal-Language*	97.2 / 96.4	98.4 / 96.6	96.6 / 95.8	98.4 / 96.8
	Object-Language	97.8 / 98.8	98.8 / 99.8	98.4 / 99.2	98.8 / 99.8
	Spatial-Object	96.8 / 96.2	98.2 / 98.4	97.0 / 95.8	98.0 / 97.2
	Goal-Position	34.0 / 34.0	31.6 / 31.0	34.4 / 33.6	32.8 / 32.4
	Goal-Task	8.6 / 20.0	9.0 / 22.2	8.6 / 21.2	9.0 / 20.2
LIBERO-Plus	*object-camera*	84.34 / 92.42	88.38 / 92.42	85.35 / 91.92	90.15 / 92.17

Table 8: Additional results: before / after SR (%) for mass adjustment; SR (%) / score (0–100) for motion-aware scoring. Blue/pink marks each row's worst/best values separately for each condition, including ties.

Benchmark	Task	Baseline	DP-Cache-Fast	DP-Cache-Slow	ProbeFlow
Mass adjustment					
RoboTwin	*place a2b right*	71 / 68	74 / 59	78 / 62	68 / 62
	stack blocks three	26 / 34	26 / 31	18 / 26	28 / 31
LIBERO	*Plate Pushing*	100 / 86	100 / 90	100 / 80	100 / 88
Motion-aware scoring					
RoboTwin	*pick diverse bottles*	37 / 37.0	36 / 18.2	38 / 30.4	41 / 17.8
	pick dual bottles	49 / 48.6	49 / 25.2	46 / 36.8	50 / 21.6

Table 9 lists the RoboTwin and LIBERO mass changes in grams. In LIBERO, the yellow-and-white mug in *Mug Placement in Microwave* (including door closure) changes from 30.91 to 300 g (9.7×); the plate in *Plate Pushing* from 11.52 to 150 g; and the white porcelain mug in *Two-Mug Placement* from 16.46 to 250 g. The latter task's SR comparison appears in Table 3.

Table 9: Object masses used in the RoboTwin and LIBERO mass-adjustment comparisons (g).

Task	Object	Original (g)	Adjusted (g)
RoboTwin			
beat block hammer	Hammer	1	600
place a2b right	Bell	50	150
stack blocks three	Block	10	50
place fan	Fan	10	600
LIBERO			
Mug Placement in Microwave	Yellow-and-white mug	30.91	300
Plate Pushing	Plate	11.52	150
Two-Mug Placement	White porcelain mug	16.46	250

D Practical Guidelines for Assessing Acceleration Gains

The repairs above motivate four practical recommendations for assessing whether task-specific gains under acceleration reflect valid task completion under fair comparison conditions.

Advice #1. Make robots follow human instructions.

Policies should follow instructions rather than rely on memorized scenes. Align expert demonstrations and success checks with the requested actions and outcomes. State necessary constraints explicitly and remove checks that impose unstated requirements or reward unintended behavior. Document repairs to existing benchmarks and enforce this consistency when building new ones.

Advice #2. Validate the task the policy actually receives.

Inspect the delivered prompt and the fully initialized scene, since loading and perturbation mechanisms can change the task presented to the policy. Check for incorrect or irrelevant text in instructions, mismatches between the scene and its initial state, and unavailable operations. Verify that initial spatial relations match the task description and that success is not already satisfied before execution.

Advice #3. Make baseline and accelerated runs use matched task instances.

A shared seed does not establish that two runs instantiate the same task. Control all random sources and save the information needed to restore the scene and its scoring targets. Verify that baseline and accelerated runs receive matching instructions, initial states, and initial model inputs, with identical success criteria. Test state restoration and action replay separately to establish which aspects of an episode can be reproduced.

Advice #4. Beyond success rate: motion quality and completion time.

Report motion jitter and end-to-end completion time alongside success rate. Distinguish policy-output variation from executed-motion jerk, using a consistent time base for each. Include both inference and robot motion in completion time, and report failures and timeouts so early termination is not mistaken for efficiency.

The bugs and design limitations uncovered by our analysis of anomalous gains provide diagnostic leads for other benchmarks and tasks: examine whether similar mechanisms occur and account for apparent improvements under acceleration. Where such issues are found, compare the baseline and accelerated policies on matched task instances before and after repairing the bugs or strengthening the evaluation criteria to address the limitations. Use trajectory evidence to examine both failures that become successes and successes that become failures across task types, object layouts, and operation stages. Gains that persist under these checks provide stronger evidence that an acceleration method is well suited to the task, whereas gains that disappear or reverse suggest that the apparent task affinity may stem from benchmark flaws.

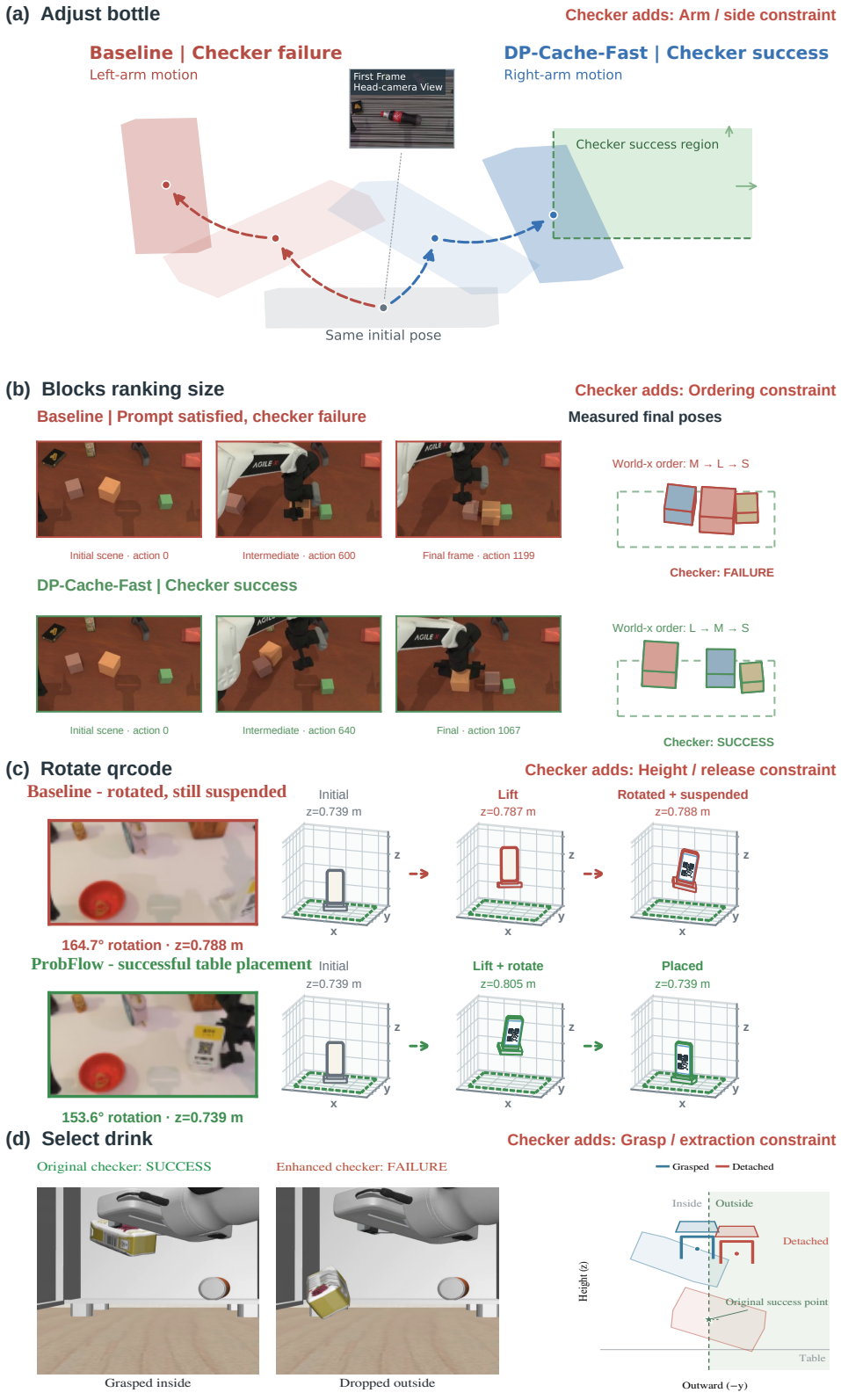


Figure 9: Visual examples of benchmark issues. Bug details and repairs are provided in Table 6.