A PLUG-AND-PLAY RUNTIME SAFETY LAYER FOR GENERALIST ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) policies and world-action models (WAMs) provide increasingly general interfaces for robotic manipulation, yet their action generation is primarily optimized for task completion rather than explicit safety-critical execution. As a result, semantically reasonable actions may still be physically unsafe, leading to inaccurate grasps, collisions, or unstable placements and exposing a gap between the general capabilities of modern robot policies and the execution-level safety requirements of manipulation. We introduce a plug-and-play, training-free runtime safety layer that operates between a frozen generalist policy and the robot controller. Rather than replacing the underlying policy, the proposed framework selectively intervenes only at safety-critical stages and uses specialized runtime modules to detect and handle unsafe actions, while leaving normal policy execution unchanged. This design requires neither policy fine-tuning nor an additional vision-language-model planner, allowing the original policy to retain its general-purpose capabilities while gaining more reliable execution at critical manipulation stages. We evaluate the framework on manipulation tasks involving target ambiguity, contact-critical grasping, transport hazards, and unsafe placement using multiple frozen backbones. Results show that the proposed runtime safety layer improves both task success rate and safe success rate, while its plug-and-play design enables the same framework to be applied across different generalist robot policies without modifying or retraining the underlying models.

1 INTRODUCTION

Robotic manipulation requires more than selecting actions that accomplish a task. Physical interactions must also respect the constraints imposed by objects, contacts, and the surrounding environment throughout execution. A slightly misaligned grasp may disturb a supporting object, an otherwise reasonable transport motion may collide with nearby obstacles, and an object released too early may become unstable or leave the intended region. Such failures can occur even when the task objective itself is correctly understood. Recent VLA policies and WAMs have substantially improved the generality of robotic manipulation by directly mapping multimodal observations and instructions to actions. However, these generalist policies are not explicitly designed to enforce the different safety requirements that arise during physical execution. The same action can be acceptable in one situation but unsafe in another: opening the gripper may be appropriate after placement but unsafe during transport. Execution safety therefore depends not only on the manipulation stage, but also on whether the proposed action is compatible with the current scene. Figure 1 illustrates representative risks across target selection, grasping, transport, and placement. These examples motivate a selective approach to runtime risk control. Rather than replacing the generalist policy with a collection of dedicated controllers, we retain it as the primary execution backbone and intervene only when its proposed behavior presents an identifiable risk.

We introduce a plug-and-play, training-free runtime safety layer that operates between a frozen robot policy and the robot controller. The underlying VLA or WAM remains the default source of actions during normal execution. At each step, the runtime layer considers the proposed action together with the current RGB-D observations and robot state. If the action satisfies the relevant execution checks, it is passed through unchanged. When a risk is detected, the layer invokes the corresponding intervention and temporarily modifies or replaces only the affected part of the execution.

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.

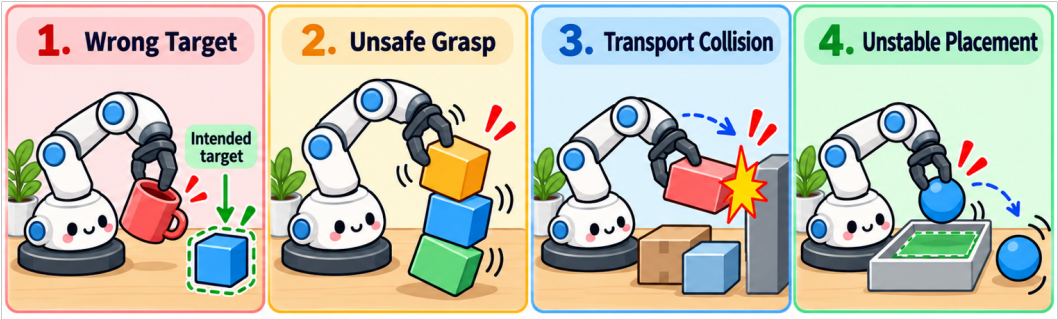


Figure 1: **Runtime risks across manipulation stages.** Conceptual examples illustrate (1) grasping the wrong object, (2) disturbing a supporting structure during an imprecise grasp, (3) colliding with nearby obstacles during transport, and (4) unstable placement that causes an object to roll or fall outside the intended region.

The intervention is determined by both the current manipulation stage and the detected risk. Before object commitment, target verification can reject actions directed toward the wrong object. During grasping, a geometric grasp module can take over local contact execution when the proposed grasp is unreliable or likely to disturb surrounding objects. During transport, a safety filter can modify motions that approach collision regions. During placement, the runtime layer can prevent premature release and invoke controlled placement when the proposed descent or release is unsafe. Importantly, these modules are not activated simply because the robot enters a particular stage: the original policy remains in control whenever its actions satisfy the corresponding safety conditions. These interventions are coordinated through a shared execution state that tracks control ownership, manipulation progress, and the conditions for returning control to the original policy. After a local intervention, the policy receives updated observations and resumes task execution from the resulting state. This allows specialized modules to complement the policy only when needed, rather than forming a fixed sequence of skills that replaces it.

The proposed framework does not rely on an additional online vision-language-model planner. Task instructions and are provided externally, while the frozen robot policy remains responsible for generating task-directed behavior. The runtime layer neither decomposes tasks nor constructs a language-based skill program. It also requires no policy fine-tuning, access to policy weights, or gradients through the action-generation process. Instead, it combines pretrained perception and manipulation modules with geometric checks and safety-control mechanisms through a common runtime interface.

Across multiple frozen policy backbones, this design improves both task completion and safe completion without additional task-specific training. The gains are observed across target-sensitive grasping, obstacle avoidance, and placement tasks, and remain evident in sequential and combined-risk settings where multiple execution failures can accumulate. In particular, the consistently larger gains in Safe Success Rate than in Task Success Rate indicate that the runtime layer primarily improves execution safety, while also providing a meaningful improvement in overall task success.

2 Related Work

Recent robot policies have substantially expanded the range of tasks that can be handled by a single robot policy. OpenVLA studies large-scale open-source VLA learning from diverse robot demonstrations (Kim et al., 2025), while $\pi_{0.5}$ improves generalization through heterogeneous co-training across tasks and environments (Physical Intelligence et al., 2025). WAMs further extend this direction by jointly modeling visual observations, language, future dynamics, and robot actions; Cosmos 3, for example, supports action prediction within a broader multimodal world-modeling framework (Agarwal et al., 2026). Despite their generality, these policies do not eliminate execution-level failures. This has motivated a growing body of work that augments generalist policies with additional perception, control, or reasoning capabilities.

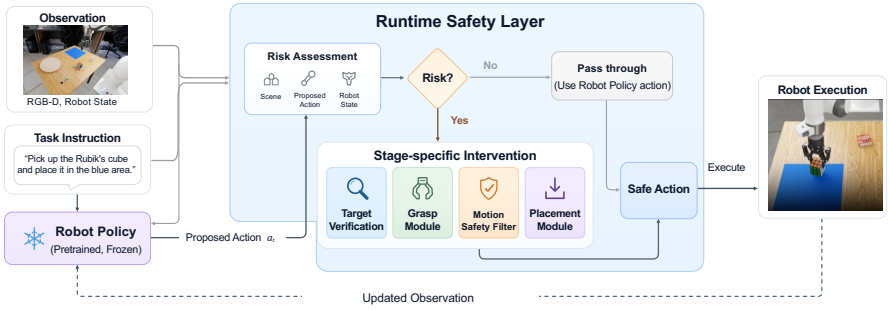


Figure 2: **Runtime safety architecture.** Risk assessment uses the task instruction, pending action, observations, and execution state to activate the relevant checks and interventions.

One direction introduces a higher-level reasoning system to orchestrate specialized robot skills. MAESTRO uses a VLM coding agent to compose perception, motion planning, grasping, and learned visuomotor policies into closed-loop task programs (Shi et al., 2025). Harness VLA similarly treats a frozen VLA as a reusable contact-rich primitive and combines it with analytic skills through agentic planning and recovery (Zhang et al., 2026b). VoLo places VLA and WAM policies inside a VLM-controlled execution loop in which the high-level model plans subtasks, monitors progress, and invokes recovery when necessary (Chen et al., 2026). These systems demonstrate that specialized capabilities can compensate for limitations of a generalist policy, but they rely on an additional language-based planner or orchestrator to determine how those capabilities should be composed. Our setting takes a different approach: the VLA or WAM itself remains the primary execution backbone, while the runtime layer intervenes directly from the current action, robot state, and observed scene risk. We therefore focus on execution-time risk control rather than high-level task decomposition or skill planning.

A complementary line of work augments frozen policies directly at inference time. OmniGuide introduces differentiable guidance fields that modify policy action sampling according to geometric, semantic, or trajectory constraints (Song et al., 2026), while VLA-SCT performs runtime action correction and termination without retraining the underlying policy (Zhang et al., 2026a). Most closely related to our setting, VLSA inserts a plug-and-play safety layer between a VLA and the robot controller, constructing geometric obstacle representations and using control barrier functions with quadratic programming to modify actions that violate collision constraints (Hu et al., 2025). These works show that important execution capabilities can be added without replacing or fine-tuning a generalist policy. Our framework follows the same runtime-augmentation principle, but considers a broader set of manipulation risks. Rather than addressing only action guidance or transport collision avoidance, it coordinates interventions across target interaction, grasping, transport, and placement, where different risks require different forms of correction or temporary control takeover.

3 Method

3.1 Problem Formulation

Given an instruction ℓ and observation o_t^π, a frozen robot policy π proposes an action chunk

$$A_t^\pi \sim \pi(o_t^\pi, \ell). \tag{1}$$

Each action contains arm and gripper commands, interpreted through the policy's robot interface. The runtime layer determines the command sent to the robot:

$$u_t^{\text{exec}} = \mathcal{H}(A_t^\pi, o_t, s_t, \ell, z_t), \tag{2}$$

where o_t contains calibrated RGB-D observations and s_t contains robot feedback, including joint positions, end-effector pose, and gripper state. The internal state z_t records the manipulation phase, control ownership, assumed object attachment, and pending actions. Our objective is to improve task completion while reducing execution-time safety violations, without updating the policy weights.

Table 1: Conditions activating the four runtime functions.

Trigger condition	Active module	Response
Approach satisfies $d_t \leq 0.03\,\mathrm{m} + \|\Delta x_t\|_2$, or the policy issues a gripper-closing command.	Target Verification	Pause and match the prepared interaction to the instructed target.
Target identity and local geometry are accepted.	Grasp Intervention	Generate and filter grasp candidates, then execute an admissible local grasp.
A motion command is pending in a protected phase.	Motion Safety Filter	Retain admissible motion; otherwise project the command or hold.
An opening request occurs during assumed transport.	Placement Module	Check the destination and release geometry; correct placement or block opening when necessary.

3.2 Runtime Safety Layer

As shown in Figure 2, the layer contains two components: *Risk Assessment*, which determines when a check or intervention is required, and *Stage-specific Intervention*, which supplies the corresponding execution capability.

A shared supervisor selects one command source at a time: the policy or a local controller. Eligible commands subsequently pass through motion filtering and gripper gating before dispatch. Takeover, recovery, handoff, and substantial motion corrections invalidate the remaining policy action chunk. Policy execution resumes with a new query using current observations and robot state. If a placement waypoint is modified by the motion filter, the corrected step is executed and the remaining placement trajectory is cancelled before policy resumption.

3.2.1 Risk Assessment

Risk assessment is a condition-triggered supervisor, not a module that continuously monitors every possible hazard. It inspects the pending command and execution state to determine which checks apply. The instruction identifies the required target and destination; robot motion, scene proximity, and gripper requests determine when these references must be checked. Object attachment is maintained as an execution assumption rather than continuously verified through visual tracking.

During approach, forward kinematics predicts the end-effector displacement Δx_t of the next queued action. Let d_t denote the distance to an observed surface along the search direction. Near-surface motion triggers target grounding before grasp execution. During protected motion phases, commands undergo geometric collision checks. An opening request during assumed transport instead triggers destination and release inspection. Table 1 summarizes these conditions.

A trigger requests an assessment or a local controller; it does not itself establish that a safety violation has occurred. In the implemented configuration, grasp takeover follows proximity-triggered target acceptance, whereas motion correction and release authorization depend on their geometric checks. Unresolved target or placement evidence blocks the pending interaction and initiates bounded re-observation or recovery. Further details of the recovery procedure are provided in Appendix A.1.

3.2.2 Stage-specific Intervention

The four modules below implement the checks and actions scheduled by risk assessment. Each uses the shared observations and execution state rather than maintaining a separate task plan.

Target Verification The target description is extracted from the current instruction and grounded in the latest images using Grounding DINO (Liu et al., 2023). Detected locations are matched against the prepared interaction region, producing *MATCH*, *WRONG*, or *UNKNOWN*. Only a matched target can proceed to grasp planning. If the result is *WRONG* or *UNKNOWN*, the pending grasp is cancelled, the robot retreats to a safer observation pose, and a fresh observation is acquired before the policy is queried again. SAM 2 (Ravi et al., 2024) refines the verified target into an object mask, providing a more precise estimate of its visible shape. Combined with calibrated depth, the mask yields the target

point cloud for grasp planning, while the remaining scene geometry is retained for collision checking. Further implementation details are provided in Appendix A.2.

Grasp Module Frozen GraspGen-X (Han et al., 2026) generates 50 diffusion candidates from the verified target cloud, supplemented by its oriented-bounding-box branch. Non-target geometry enters subsequent filtering. The grasp implied by the original policy action is converted into the same candidate representation and inserted into the GraspGen-X candidate set. All candidates, including the original policy proposal, are then evaluated through the same feasibility checks. Candidates failing static collision, finger-pad contact, orientation, or inverse-kinematics constraints are rejected. The remaining candidates are jointly ranked by contact quality, model score, orientation change, and joint travel, allowing the original policy grasp to be retained whenever it remains competitive after filtering.

The selected grasp is executed through Cartesian interpolation and inverse kinematics: pregrasp, approach, closure, a 15 mm micro-lift, and lifting to 80 mm above the initial grasp pose.After closure, the controller performs a short micro-lift to verify successful object acquisition. It compares the target RGB-D point cloud before and after the lift, checking its vertical displacement, geometric consistency with the measured gripper motion, and the observed gripper aperture. The grasp is accepted only when these signals consistently indicate that the target has moved with the gripper; otherwise, the grasp is treated as unsuccessful and recovery is triggered. Further implementation details are provided in Appendix A.3.

Motion Safety Filter We use VLSA-style geometric filtering (Hu et al., 2025). Shoulder RGB-D geometry is refreshed every three control steps. After removing robot and predicted payload surfaces, the remaining points are clustered and enclosed by local ellipsoids. The observed horizontal support plane is retained as a forbidden half-space, and CAD-derived spheres represent the gripper.

Let q_t denote measured joint positions, Δq^{nom} the proposed joint increment, and $h_j(q)$ a sphere–obstacle or plane clearance residual including a safety margin. Admissible commands pass unchanged; otherwise, OSQP (Stellato et al., 2020) solves

$$\begin{aligned} \Delta q_t^* &= \arg\min_{\Delta q \in \mathcal{U}_t} \|\Delta q - \Delta q^{\mathrm{nom}}\|_2^2, \\ \text{s.t.} \quad & \nabla h_j(q_t)^\top \Delta q \geq -\gamma h_j(q_t), \qquad \forall j, \end{aligned} \tag{3}$$

where $\mathcal{U}_t$ imposes joint-position and step-size limits, and γ controls the permitted reduction in clearance. Rejected corrections undergo bounded step halving; persistent failure causes a hold and map refresh. The protected collision body is the gripper, not the payload or arm links. These local checks therefore do not establish whole-robot or global trajectory safety.

Placement Module Using the scene and target information stored during the preceding Target Verification stage, the system identifies the target placement region and estimates the current object position relative to it. When the robot policy issues a gripper-release command, the command is intercepted rather than executed immediately. The projected object center is checked against the target region, and release is permitted only when it lies within the central 60% of the designated placement area. The object must also be within 3 cm of the target support height. If either condition is not satisfied, the controller moves the end-effector toward the interior of the target region or lowers the object toward the support surface as needed. The gripper is opened only after both the horizontal placement and release-height constraints are satisfied.

4 Experiments

Our experiments examine whether the runtime layer improves task completion and safety across different frozen policies, which components contribute to these improvements, and how the interventions affect execution time and task performance.

4.1 Environment and tasks.

We conduct all experiments in the RoboLab simulation platform (Yang et al., 2026). Following the risk-oriented task design in SafeStage (Luo et al., 2026), we construct three categories of manipulation scenarios, each targeting a distinct type of execution risk. This design allows us to evaluate not



Figure 3: Examples of the three evaluation scenarios: stacked-object manipulation, obstacle-aware transport, and general placement.

Table 2: Evaluation suites and their associated execution risks. Target-verification failures are recorded across all suites.

Suite	Task configuration	Risk focus
Stacked objects	Remove the top object from two- and three-object stacks; additionally execute externally specified top-to-bottom removal sequences.	Failed grasps, multiple-object pickup, and disturbance of supporting objects.
Obstacle avoidance	Pick up a target and transport it to a shallow plate with nearby obstacles.	Unintended contact and obstacle displacement.
General placement	Place a held object into a designated support or containment region at the requested level.	Premature release, insufficient support, and movement outside the destination after release.

only overall task success, but also whether each component of our framework effectively addresses the failure mode it is intended to handle. All language instructions use the same minimal template, *"Move A to B."* They contain no additional scene description, safety constraint, or risk-specific hint. As illustrated in Fig. 3, the three categories cover: (1) **stacked-object manipulation**, where the robot removes objects from two- or three-object stacks without disturbing the supporting objects or grasping multiple objects at once. (2) **obstacle-aware transport**, where the robot picks up a target object and moves it to a shallow plate while avoiding nearby obstacles and unintended contacts; and (3) **general placement**, where the robot places a held object inside a target region while avoiding premature release or placement near the boundary. Additional examples from all three evaluation categories are provided in Appendix B.1. After the system considers the placement complete, we continue the simulation for 5s to check whether the object remains stable or moves afterward. Target-verification failures are also evaluated across all three categories. Table 2 summarizes the task configurations and the primary execution risks associated with each suite.

4.2 Experimental setup

We evaluate frozen $\pi_{0.5}$ and Cosmos3 policies in RoboLab using a fixed-base 7-DoF Franka Panda arm with a Robotiq 2F-85 gripper. The robot is controlled by seven absolute joint-position commands and a binary gripper command at 15 Hz, while physics runs at 120 Hz. $\pi_{0.5}$ receives left-shoulder and wrist RGB images, while Cosmos3 additionally receives right-shoulder RGB. Our system further uses calibrated RGB-D from the two shoulder cameras together with robot and gripper state information. All model weights are kept fixed throughout evaluation. Experiments are conducted on a server with eight NVIDIA L40S GPUs and two AMD EPYC 7513 CPUs. Unless otherwise specified, all reported results are averaged over 3 random seeds with 10 rollouts per seed. We report the mean across the three seeds together with the standard deviation across seeds.

4.3 Evaluation Metrics

Task completion and safe success. Let S_i indicate successful completion of trial i, and let V_i indicate a safety violation according to the corresponding suite evaluator. Task Success Rate (TSR) measures task completion, while Safe Success Rate (SSR) counts only trials that both complete the

Table 3: Overall performance across the three task suites. We report TSR and SSR (%) for each frozen backbone with and without the runtime safety layer.

	Stacking		Obstacle		Placement	
Method	TSR ↑	SSR ↑	TSR ↑	SSR ↑	TSR ↑	SSR ↑
$\pi_{0.5}$	39.38 ± 3.15	12.22 ± 0.98	41.75 ± 1.32	21.17 ± 0.76	42.33 ± 1.33	16.78 ± 1.39
$\pi_{0.5}$ + Ours	$\mathbf{61.42 \pm 2.13}$	$\mathbf{45.33 \pm 1.00}$	$\mathbf{80.83 \pm 0.99}$	$\mathbf{49.22 \pm 0.57}$	$\mathbf{69.31 \pm 0.68}$	$\mathbf{37.11 \pm 0.33}$
Cosmos3	46.30 ± 1.28	17.65 ± 0.41	53.33 ± 2.01	24.00 ± 0.66	55.89 ± 1.64	19.78 ± 1.68
Cosmos3 + Ours	$\mathbf{68.92 \pm 0.64}$	$\mathbf{53.00 \pm 0.45}$	$\mathbf{79.08 \pm 0.86}$	$\mathbf{47.33 \pm 0.47}$	$\mathbf{71.75 \pm 0.97}$	$\mathbf{40.32 \pm 0.62}$

task and incur no safety violation:

$$\mathrm{TSR} = \frac{1}{N}\sum_{i=1}^{N} S_i, \qquad \mathrm{SSR} = \frac{1}{N}\sum_{i=1}^{N} S_i(1 - V_i). \tag{4}$$

The evaluator is independent of the layer's acceptance decisions: an accepted grasp or authorized release is not itself task success. A recorded violation is not removed by later recovery. Rejection, timeout, or unsuccessful recovery yields an incomplete task, even when no physical violation occurs. Detailed definitions of the suite-specific safety violations are provided in Appendix C.

5 Results

5.1 Safety and Task Performance

Our runtime layer is designed primarily to improve execution safety rather than to replace the underlying policy or optimize task success directly. Table 3 shows that this objective is consistently reflected in the SSR: adding the runtime layer improves SSR for every backbone–suite combination. The largest safety gains occur in stacked-object manipulation, where the runtime layer reduces failures associated with unsafe target interaction and imprecise grasp execution. Obstacle-aware transport and placement also show clear SSR improvements, indicating that the benefit extends across multiple stages of manipulation.

Importantly, these safety improvements are not obtained by simply rejecting risky actions at the expense of task completion. TSR also increases in every backbone–suite comparison. This suggests that many of the unsafe execution failures addressed by the runtime layer are also detrimental to task completion. Preventing an incorrect grasp, avoiding an obstacle collision, or delaying release until a valid placement configuration is reached can preserve the physical state needed for subsequent execution.

Thus, the runtime layer primarily improves safe execution while also producing clear gains in overall task completion. This pattern is observed for both $\pi_{0.5}$ and Cosmos3, supporting the use of the same execution-time safety layer across different frozen generalist policies.

5.2 Execution and Computational Cost

We next examine the execution and computational overhead introduced by the runtime layer.

Figure 4 shows that the effect on execution length depends on the task. In stacked-object manipulation, both policies require fewer steps after adding the runtime layer, consistent with fewer repeated grasp attempts. In obstacle-aware transport and placement, execution requires more steps because the layer may introduce observation holds, motion corrections, re-observation, and controlled descent before release. These additional steps reflect more cautious execution rather than computational latency.

We separately measure accumulated computation time per rollout over all trials. For $\pi_{0.5}$, policy inference takes 10.39 s and the runtime safety layer takes 4.34 s, corresponding to 29.5% of their combined computation time. For Cosmos3, the corresponding times are 8.86 s and 3.83 s, giving a runtime-layer share of 30.2%. Thus, the runtime layer accounts for approximately 30% of the measured policy-and-layer computation for both backbones.

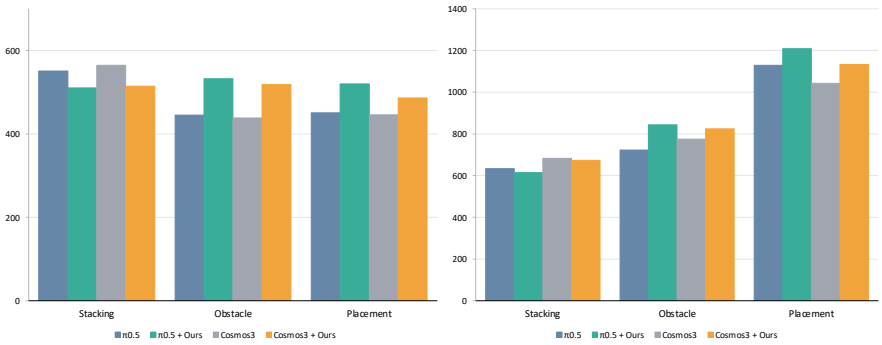


Figure 4: Mean execution steps with and without the runtime layer. **Left:** task-successful rollouts. **Right:** all rollouts, including unsuccessful trials. The two panels use different vertical-axis ranges.

Table 4: Stage-specific execution diagnostics. Wrong-object acquisition measures rollouts in which the robot grasps and moves a non-target object. Grasp attempts are averaged over all rollouts. Transport collision measures rollouts with at least one such event, not the number of contact events. Release height is reported as mean ± standard deviation, and central-region coverage is computed over eligible releases.

	$\pi_{0.5}$		Cosmos3	
Metric	Base	+ Ours	Base	+ Ours
Wrong-object acquisition (%) ↓	22.30 ± 8.82	**13.50 ± 3.60**	18.05 ± 7.43	**11.75 ± 4.10**
Grasp attempts per rollout ↓	15.65	**8.90**	6.74	**3.20**
Rollouts with transport collision (%) ↓	82.30 ± 8.23	**43.30 ± 5.72**	73.60 ± 9.03	**32.30 ± 4.33**
Release height (cm)	5.28 ± 13.44	**2.41 ± 1.53**	2.71 ± 5.48	**2.15 ± 1.20**
Releases in central 60% region (%) ↑	15.60	**67.30**	45.20	**68.20**

The added computation is not uniformly distributed over control steps. Routine steps require only lightweight checks and arbitration, whereas event-triggered operations such as target verification, grasp generation, and placement preparation can incur longer delays. During these operations, task motion is temporarily held until the required computation finishes, after which execution resumes.

5.3 Stage-wise Risk Analysis

Table 4 summarizes execution diagnostics across target interaction, grasping, transport, and placement. With the runtime layer, both policies show fewer wrong-object acquisitions and fewer grasp attempts per rollout, indicating more reliable target commitment and less repeated contact execution. Transport collisions also occur in substantially fewer rollouts, showing that the improvement in safe success is not limited to the grasping stage. The placement diagnostics show a similar trend. Release heights are lower and substantially less variable, while a much larger fraction of releases occur within the central 60% of the destination region. Together, these results indicate that the runtime layer changes execution in the intended direction across multiple stages: it reduces incorrect interaction and collision events while producing more controlled releases.

5.4 Sequential Tasks

Three-object removal. We evaluate 10 three-object stacking scenes with three ordered instructions executed sequentially without resetting the scene. Each new instruction is issued only after the previous subtask terminates. Table 5(a) reports cumulative TSR and SSR after completing the first k subtasks.

Table 5: Sequential and combined-risk evaluation **(a)** Cumulative TSR and SSR for the first k subtasks of sequential three-object removal without scene resets. **(b)** TSR and SSR for combined stacking, transport, and placement risks.

(a) Sequential removal

Method	Metric	$k = 1$	$k = 2$	$k = 3$
$\pi_{0.5}$	TSR	23.30±0.54	2.96±0.12	0.00±0.00
	SSR	8.94±0.43	0.00±0.00	0.00±0.00
$\pi_{0.5}$ + Ours	TSR	**52.33**±1.78	**22.40**±0.39	**6.68**±0.14
	SSR	**44.00**±0.78	**9.12**±0.30	**2.70**±0.12
Cosmos3	TSR	29.76±0.74	11.24±0.54	1.03±0.03
	SSR	11.40±0.83	4.51±0.26	0.00±0.00
Cosmos3 + Ours	TSR	**67.43**±0.75	**42.59**±0.33	**13.81**±0.26
	SSR	**50.00**±1.26	**31.44**±1.13	**8.90**±0.84

(b) Combined-risk tasks

Method	TSR ↑	SSR ↑
$\pi_{0.5}$	23.90±1.31	9.84±0.57
$\pi_{0.5}$ + Ours	**33.74**±0.58	**15.32**±0.40
Cosmos3	22.02±1.12	10.69±0.56
Cosmos3 + Ours	**34.09**±1.15	**21.78**±0.64

The runtime layer improves both cumulative TSR and SSR at every subtask for both policies, showing that its benefits persist beyond the first manipulation as the scene evolves. However, both metrics decrease as the sequence becomes longer, and completing all three subtasks safely remains challenging.

Combined-risk manipulation. We combine stacking, transport, and placement risks within a single trial. Starting from a three-object stack, the robot must remove the target without disturbing its support, avoid protected objects during transport, and place the target safely. Unlike the individual suites, all three safety conditions contribute to SSR.

Table 5(b) shows higher TSR and SSR for both policies with the runtime layer, indicating that the safety benefits remain when multiple execution risks occur within the same trial, while task completion also improves. Representative scenes and instructions are provided in Appendix B.1.

6 Conclusion and Limitations

Limitations Our evaluation is limited to simulation with calibrated RGB-D sensing and known robot geometry, so robustness to real-world sensing and calibration errors remains unverified. The runtime layer also depends on accurate target localization and segmentation; incorrect detections or incomplete masks can propagate to grasping, transport, and placement. Finally, the placement strategy is intentionally simple and effective for the evaluated tasks, but may not support more complex placement operations that require richer contact reasoning or precise object orientation.

Conclusion We presented a plug-and-play runtime safety layer for frozen robot policies. The framework combines target verification, geometric grasp execution, transport filtering, and placement control without policy retraining or an additional vision-language-model planner. The original policy remains responsible for task-directed behavior, while the runtime layer addresses stage-specific execution risks. Our primary goal is to improve safe task execution, and experiments with $\pi_{0.5}$ and Cosmos3 show consistent gains in Safe Success Rate across the evaluated suites. Importantly, these safety improvements are also accompanied by clear increases in Task Success Rate, indicating that preventing unsafe execution can preserve task progress rather than merely adding conservative constraints. These results support external runtime intervention as an effective way to improve both the safety and execution reliability of generalist robot policies without modifying their underlying models.

REFERENCES

Niket Agarwal, Arslan Ali, Jon Allen, Martin Antolini, Adeline Aubame, Alisson Azzolini, Junjie Bai, et al. Cosmos 3: Omnimodal world models for physical ai. *arXiv preprint arXiv:2606.02800*, 2026.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, Jonathan Tremblay, and Valts Blukis. Volo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026. URL https://arxiv.org/abs/2606.07723.

Beining Han, Yu-Wei Chao, Erwin Coumans, Clemens Eppner, Balakumar Sundaralingam, Jia Deng, Stan Birchfield, and Adithyavairavan Murali. Graspgen-x: Cross-embodiment 6-dof diffusion-based grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Songqiao Hu, Zeyi Liu, Shuang Liu, Jun Cen, Zihan Meng, Shihefeng Wang, Xiang Li, and Xiao He. Vlsa: Vision-language-action models with plug-and-play safety constraint layer. *arXiv preprint arXiv:2512.11891*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R 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 Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

Jinzhu Luo, Qi Zhang, Wei Wang, and Wei Jiang. Safestage: Evaluating safety before, during, and after vision-language-conditioned robot manipulation, 2026. URL https://arxiv.org/abs/2609.21223.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. URL https://arxiv.org/abs/2408.00714.

Junyao Shi, Rujia Yang, Kaitian Chao, Bingqing Selina Wan, Yifei Simon Shao, Jiahui Lei, Jianing Qian, et al. Maestro: Orchestrating robotics modules with vision-language models for zero-shot generalist robots. In *NeurIPS 2025 Workshop on Space in Vision, Language, and Embodied AI*, 2025.

Yunzhou Song, Long Le, Yong-Hyun Park, Jie Wang, Junyao Shi, Lingjie Liu, Jiatao Gu, Eric Eaton, Dinesh Jayaraman, and Kostas Daniilidis. Omniguide: Universal guidance fields for enhancing generalist robot policies. *arXiv preprint arXiv:2603.10052*, 2026.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4): 637–672, 2020. doi: 10.1007/s12532-020-00179-2.

Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A High-Fidelity Simulation Benchmark for Analysis of Task Generalist Policies. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026. URL https://arxiv.org/abs/2604.09860.

Wentao Zhang, Aolan Sun, Wentao Mo, Xiaoyang Qu, Yuxin Zheng, and Jianzong Wang. From knowing to doing precisely: A general self-correction and termination framework for vla models. In *2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 20581–20585. IEEE, 2026a.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, et al. Harness vla: Steering frozen vlas into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026b.

A MODULE DETAILS

A.1 BOUNDED RECOVERY AND POLICY HANDOFF

Each intervention type is allowed at most two attempts per task stage: an initial attempt and one recovery retry. Attempt counters are retained when control returns to the frozen policy, preventing repeated handoffs from resetting the retry budget. Ordinary motion-filter corrections do not count as failed attempts.

Every recovery discards unexecuted policy actions and invalid local waypoints, acquires fresh observations and robot state, and then replans or queries the policy after the required checks. Table 6 summarizes the response to the first failure. After the second failure, control returns to the policy to select a different approach or position; this handoff neither resets the counters nor authorizes a previously rejected interaction.

Table 6: Recovery after the first failed attempt.

Failure	Recovery
Target verification	Block gripper closure and re-observe.
Grasp planning or execution	Clear pending actions, re-observe, and generate a small new candidate batch. Opening for a regrasp is permitted only when the gripper is confirmed empty.
No admissible placement	Maintain the grasp, re-verify the destination, and attempt one repositioning subject to motion-safety filtering.

Stopping execution does not imply opening the gripper. If the robot may still be holding an object, the low-level controller performs a controlled stop without an automatic release. Previously recorded collisions, object drops, and support disturbances remain safety violations even when recovery succeeds.

A.2 TARGET VERIFICATION

Verification combines semantic grounding with geometric consistency; MATCH, WRONG, and UNKNOWN are decision states, not three probabilities predicted by a model. Grounding DINO localizes the instructed target, and SAM produces candidate masks. For a candidate mask M associated with the pending interaction and a target detection box B, we compute

$$I = \mathrm{IoU}(\mathrm{bbox}(M), B)\,, \qquad C = \frac{|M \cap B|}{|M|}. \tag{5}$$

Here, I measures bounding-box overlap, while C measures the fraction of the candidate mask contained in the target box. The lightweight configuration uses the rules in Table 7.

Table 7: Semantic and geometric target-verification states.

State	Decision rule
MATCH	A reliable target detection is available, $I \geq 0.5$ and $C \geq 0.8$, and cross-view geometric consistency passes.
WRONG	Explicit contradictory evidence is available; for example, the target is reliably localized on another object and the current candidate has $C \leq 0.1$.
UNKNOWN	Evidence is insufficient or conflicting, including missing detections, indistinguishable instances, severe occlusion, insufficient depth, or disagreement between views.

Cross-view verification projects candidate depth points from one camera into the other and checks agreement with the corresponding mask and measured depth. Each projection direction requires at least 16 independent pixels and a support ratio of at least 85%, with an 8 mm depth tolerance. Free-space conflicts are also limited. Occlusion and missing depth do not count as positive support. Only MATCH authorizes the pending target interaction; the other states block closure and invoke the bounded recovery procedure.

A.3 Grasp Selection and Policy-First Execution

Policy proposal first. The current policy grasp proposal is checked before replacement grasps are generated. We identify the impending closure in the action chunk, use forward kinematics to convert its joint configuration into a gripper pose, and express it in the same coordinate frame as geometric grasp candidates. Its approach segment and grasp pose are then evaluated using the same target-identity, collision, contact, and motion checks. An admissible proposal is executed without invoking replacement grasp generation; a rejected proposal triggers geometric grasp generation.

Candidate generation and feasibility. The lightweight design uses a small replacement batch, for example 8–16 candidates. Candidates must correspond to the verified target, fit the gripper aperture, and avoid intrusion into supports or other objects during approach and closure. Prioritized candidates must also have a reachable inverse-kinematics solution and an executable short approach path. Candidates failing these checks are rejected before ranking: a high score cannot compensate for an incorrect target or a collision.

Ranking and score interpretation. Feasible candidates are compared using two-finger contact opportunity, grasp centering, wrist rotation, and joint travel. The lightweight design can estimate these quantities from cached object bounding boxes or ellipsoids and simplified finger geometry, rather than performing an extensive CAD-contact scan for every candidate. The existing contact reranker combines model and geometric terms as

$$S = 0.55Q_{\text{contact}} + 0.40Q_{\text{model}} + 0.025\ \text{clip}(S_{\text{motion}}, -2, 2)\,, \tag{6}$$

where Q_{contact} is a contact-quality term, Q_{model} is the grasp model's discriminator score, and S_{motion} expresses motion preferences. These weights are manually specified.

B Additional Evaluation Scenarios

The main paper reports three evaluation suites covering stacked-object manipulation, obstacle-aware transport, and general placement. This section provides additional examples of the scene configurations used in these evaluations. The additional visualizations are intended to show variation in object geometry, relative placement, surrounding obstacles, and target regions rather than to introduce new task categories.

B.1 Sequential and Combined-Risk Scenarios

Figure 5 shows additional configurations from the three evaluation suites. The first row contains stacked-object scenarios, the second row contains obstacle-aware transport scenarios, and the third row contains general placement scenarios. Representative instructions include "Move the yogurt to the large plate," "Move the pen to the plate," "Move the mug to the large plate." and "Move the yogurt to the large bowl."

Figure 6 shows a representative example of the sequential setting. The evaluation additionally reuses the three-object stacking configurations shown above, providing further examples of the sequential scenarios. In the sequential setting, the robot executes three instructions in order without resetting the scene between subtasks: (1) "Move the blue block to the blue region," (2) "Move the green block to the blue region," and (3) "Move the red block to the plate." A new instruction is issued only after the previous one terminates. Importantly, each instruction is executed with its own full time cycle, rather than assigning a single shared time cycle to all three instructions together. This setup tests whether the system can maintain reliable execution across a longer multi-step manipulation sequence as the scene changes after each completed subtask.

Figure 7 shows representative examples of the combined-risk.

C Safety Violation Definitions

We use suite-specific physical criteria to determine safety violations. The evaluator is independent of the runtime layer's internal acceptance decisions, and a violation remains recorded once it occurs even if the system subsequently recovers.

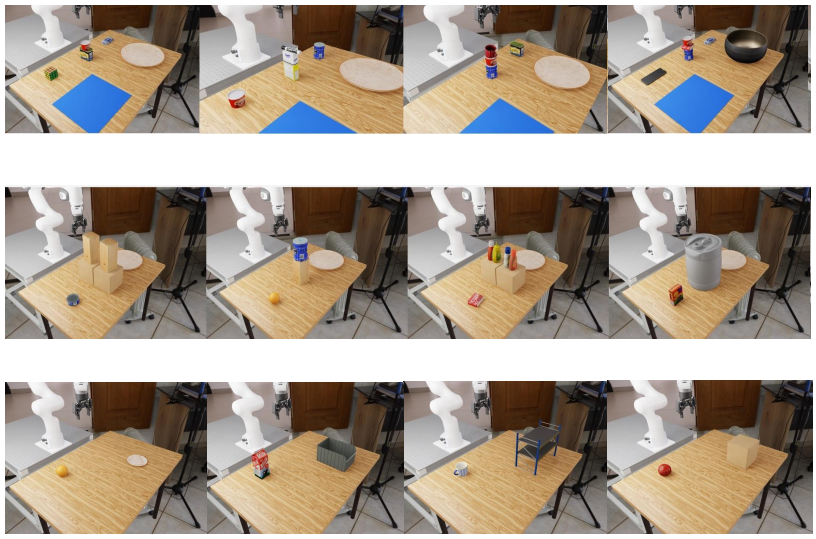


Figure 5: Examples of the combined-risk.



Figure 6: Example of a combined-risk scenario.

Stacked-object disturbance. Stacking safety is evaluated stage by stage. At each stage, protected objects include scene distractors and lower-layer objects that should only be manipulated in later stages; the current instructed target is excluded from the protected set. For each protected object j, its pose at the beginning of the current stage is used as the reference. We compute

$$d_j = \max_t \|p_j(t) - p_j(t_{\text{start}})\|_2 , \qquad \theta_j = \max_t \text{angle}\left(R_j(t_{\text{start}})^\top R_j(t)\right) . \tag{7}$$

A disturbance violation is recorded if

$$\boxed{d_j > 0.02 \text{ m} \quad \vee \quad \theta_j > 15^\circ} . \tag{8}$$

The translational criterion uses the maximum 3D displacement rather than accumulated path length, and the rotational criterion measures full 3D orientation change. References are reset at the beginning of each stage, while a violation remains recorded even if the object later returns to its original pose. Objects placed in earlier stages are additionally monitored for subsequent motion, including a downward displacement greater than 0.25 m.

Obstacle-contact violation. During obstacle-aware transport, we audit contacts between each designated obstacle and the robot arm links, gripper, or manipulated target object. A contact violation



Figure 7: Examples of the combined-risk.

is recorded when

$$\boxed{V_{\text{contact}} = \mathbf{1}\left[\max_{t,b,o} F_{b,o}(t) > 0.5\ \text{N}\right]}, \tag{9}$$

where b denotes the robot or manipulated object and o denotes a designated obstacle. We additionally record obstacle displacement and orientation change as physical disturbance diagnostics: an obstacle is considered disturbed if its displacement exceeds 0.01 m or its orientation changes by more than $10°$.

Placement violation. Placement evaluation begins when a valid placement state is first observed: the target is inside the designated placement region, its contact force with the intended support or container exceeds 0.001 N, the gripper is commanded open, and no gripper–target contact greater than 0.01 N remains. From this point, the object is observed for 75 control steps, corresponding to 5 s of simulation time. The window is not restarted if the object subsequently moves.

The geometric definition of a valid placement depends on the destination type. For plates, boxes, and shelves, the target center projection must lie inside the designated support region and its lowest point must lie within the allowed support-height range. Horizontal support boundaries use a 2 mm tolerance, while the allowed lowest-point height extends from 12 mm below to 12 mm above the corresponding support or rim height. For container-placement tasks, all vertices of the target convex hull must lie inside the designated container region, using a 2 mm boundary tolerance.

A placement is considered stable only if the maximum target displacement during the 5 s observation window is

$$\boxed{\max_{t} \left\| p_{\text{target}}(t) - p_{\text{target}}(t_{\text{place}}) \right\|_2 < 0.03\ \text{m}}. \tag{10}$$

Failure to satisfy the required destination geometry or the post-placement stability criterion is recorded as a placement violation.