TAMING VLAS UNDER ROBOT EXECUTION ERRORS: SELF-COMPENSATION AND STRESS TESTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) policies often fail when a robot's executed motion deviates from their commanded action. Such execution errors arise from the robot's mechanics and operating conditions, such as wear and payload changes. We propose self-compensating VLA, a deployment-time adaptation method that enables a VLA policy to pre-compensate for the robot's execution errors when generating commands. Without task rewards or labels, it updates the policy online using the residual between the action commanded by a VLA and the motion executed by the robot. To stress-test VLA robustness across execution conditions that are impractical to cover with physical robots alone, we introduce RoboStress, a controlled simulation benchmark. It combines established joint-level models of friction, backlash, compliance, and gravity-compensation error into seven deployment scenarios whose execution errors depend on the robot's state and motion history. On RoboStress, self-compensating VLA achieves higher average task success than both the base policies and methods that build in robustness during training. On two physical robot arms with different usage histories, it raises the average task success rate by more than 30 percentage points on each arm, and the gains extend to objects not seen in the task demonstrations.

1 INTRODUCTION

Vision-language-action (VLA) models (Zitkovich et al., 2023; Kim et al., 2024; Black et al., 2025b;a; Pertsch et al., 2025; Bjorck et al., 2025; Cheang et al., 2024; Wen et al., 2025) enable general-purpose robot manipulation, mapping camera observations and language instructions directly to robot actions across diverse tasks (Brohan et al., 2023; Octo Model Team et al., 2024). As these models move toward practical deployment, their robustness has gained increasing attention (Fei et al., 2026; Liu et al., 2025; Hancock et al., 2025; Zhang et al., 2025b; Guo et al., 2026). Existing studies (Fei et al., 2026; Liu et al., 2025; Zhang et al., 2025a) have primarily examined robustness to visual corruptions, distractors, and paraphrased instructions. Recent work has extended this analysis to action perturbations by injecting synthetic noise into a policy's output commands (Zhang et al., 2025b; Guo et al., 2026).

During physical deployment, the motion executed by a robot often differs from a VLA's action commands even without artificially injected perturbations. These execution errors stem from the robot's mechanics and operating conditions, such as wear, heavy payloads, and temperature changes (Bittencourt & Axelsson, 2014; Gaz & De Luca, 2017). Unlike the synthetic perturbations added to action commands in the previous work, these errors arise as the robot executes the commands. Although such execution errors frequently occur in physical deployments, the robustness of VLA models to them remains largely underexplored.

Motivated by this, we study the robustness of VLA models to execution errors, as illustrated in Figure 1(a). These errors vary across robots and operating conditions, calling for policy adaptation to the deployed robot. Our key insight is that the residual between the action commanded by a VLA and the motion executed by the robot, measured through proprioception, can serve as a self-supervisory signal for policy adaptation during deployment. Based on this idea, we propose a self-compensation method for VLAs, dubbed *self-compensating VLA*. Our method adapts the action policy online using this residual, without task rewards, labels, or knowledge of the error sources. 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.

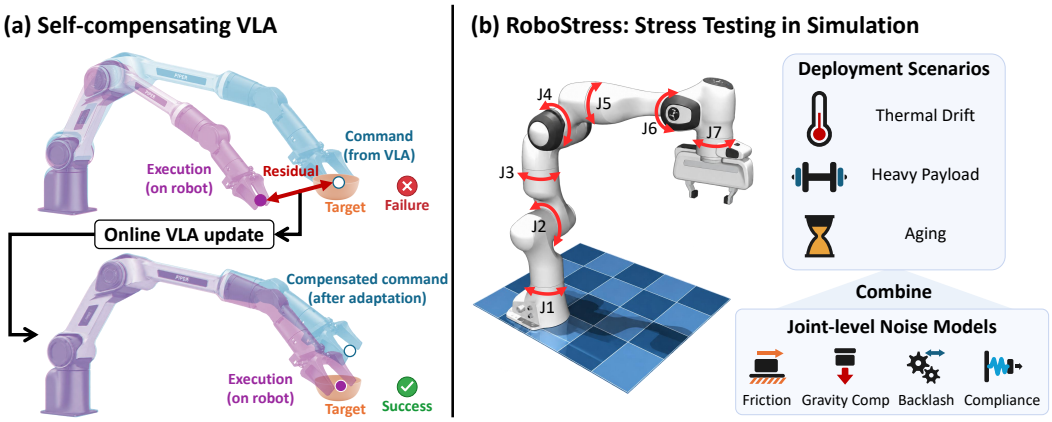

Figure 1: Overview of our study. (a) Self-compensating VLA updates the policy online using command-execution residuals and generates subsequent commands that pre-compensate for execution errors. (b) The RoboStress benchmark combines four established joint-level models into controlled deployment scenarios in simulation to stress-test VLA robustness.

updated policy then generates subsequent commands that pre-compensate for observed execution errors while leaving the robot's low-level controller unchanged.

Ideally, evaluating VLA robustness to execution errors requires a fleet of robots with diverse mechanical states, usage histories, and operating conditions. However, assembling such a fleet at scale is impractical, and varying operating conditions on physical robots is costly, with extreme settings posing a risk of hardware damage. A natural alternative is to reproduce and systematically vary these conditions in simulation. Existing simulated evaluations (Zhang et al., 2025b; Guo et al., 2026), however, rely on synthetic action perturbations that differ from execution errors encountered during physical deployments in two respects. First, real execution errors vary with both the robot's current state and its recent motion history, while the synthetic perturbations do not explicitly model such dependencies. Second, errors at individual joints affect end-effector motion differently, whereas the synthetic perturbations applied directly to end-effector commands do not capture the distinct contribution of each joint.

To address these limitations, we introduce RoboStress, a controlled simulation benchmark for stress-testing VLA robustness across diverse execution conditions, as shown in Figure 1(b). To be specific, RoboStress uses established joint-level models of friction (Canudas de Wit et al., 1995; Madsen et al., 2020), backlash (Tao & Kokotovic, 1993), compliance (Spong, 1987), and gravity-compensation error (Ma & Hollerbach, 1996; Gaz et al., 2019). Each model is applied to individual joints at the corresponding stage of the control pipeline. We combine these effects into seven deployment scenarios whose execution errors depend on the robot's state and recent motion history.

Extensive experiments on RoboStress and two physical robots validate the effectiveness of self-compensating VLA. On RoboStress, it outperforms two base policies, π_0 (Black et al., 2025b) and $\pi_{0.5}$ (Black et al., 2025a), and methods that embed robustness during training (Guo et al., 2026; Tobin et al., 2017) across all seven deployment scenarios. Beyond simulation, we evaluate self-compensating VLA on two physical Piper arms with different usage histories. For both base policies, it raises the average task success rate by more than 30 percentage points on both arms. It also improves task success with objects not seen in the task demonstrations. Our benchmark and method lay a foundation for future research on VLA robustness to robot execution errors.

2 Related Work

VLA Robustness. Previous work evaluates VLAs under input perturbations such as viewpoint, lighting, or instruction changes (Fei et al., 2026; Liu et al., 2025). Other methods improve robustness through visual input interventions (Hancock et al., 2025) or video-based planning and state representation alignment (Zhang et al., 2025a). Recent work also studies robustness to synthetic

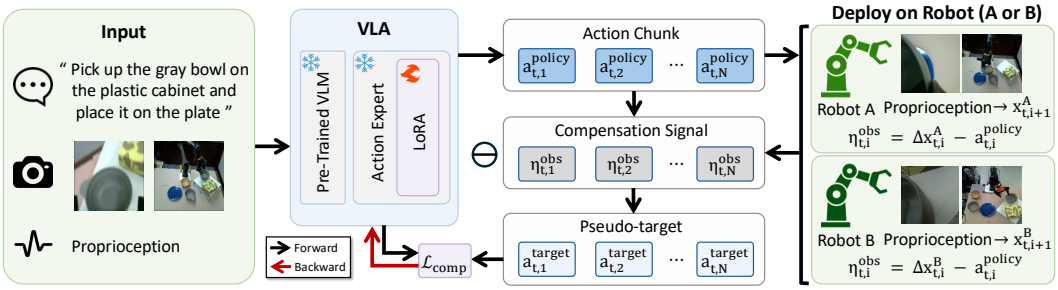

Figure 2: Overview of self-compensating VLA. A frozen VLA with trainable LoRA adapters predicts an action chunk, which the robot executes. The residual η^{obs} between the commanded and executed motion forms a pseudo-target a_t^{target}, which the compensation objective $\mathcal{L}_{\text{comp}}$ uses to update only the LoRA adapters online during deployment, leaving the base policy frozen.

perturbations applied to output end-effector commands (Zhang et al., 2025b; Guo et al., 2026). Our work instead addresses execution errors arising from robot mechanics and operating conditions.

Robot Dynamics Modeling. The joint-level models used in RoboStress are well established in robot dynamics: Coulomb-viscous and Stribeck-style friction laws (Canudas de Wit et al., 1995; Lampaert et al., 2003), hysteretic deadband models of backlash (Tao & Kokotovic, 1993), elastic-joint models of compliance (Spong, 1987; Hardeman, 2008), and gravity-compensation error from inertial-parameter and payload mismatch (Ma & Hollerbach, 1996; Gaz et al., 2019). Joint friction varies with temperature and wear (Bittencourt & Axelsson, 2014; Raviola et al., 2021), and related dynamics models have been experimentally evaluated on the Franka Panda (Gaz et al., 2019) and UR5e (Madsen et al., 2020). Whereas these models were developed for identification and control, RoboStress uses them to generate execution errors for evaluating VLA robustness.

Deployment-Time Adaptation. Earlier work adapts reinforcement learning policies during deployment through auxiliary self-supervised learning objectives (Hansen et al., 2021) or a pretrained adaptation module that infers deployment conditions from recent states and actions (Kumar et al., 2021). EVOLVE-VLA (Bai et al., 2025) and TT-VLA (Liu et al., 2026) adapt VLAs at test time using task-progress-based feedback. Our method adapts the action policy using the command-execution residual measured through proprioception, directly targeting the deployed robot's execution errors without task rewards.

Robot Control. Model-based controllers use robot dynamics and feedback to regulate motion and force in operational space (Khatib, 1987), while disturbance-observer-based methods estimate disturbances from feedback and compensate for them in the control input (Chen et al., 2016). Adaptive control handles parametric uncertainty through online parameter estimation (Slotine & Li, 1987). Learning-based control learns compensation from feedback-controller outputs (Gomi & Kawato, 1993) or adds a reward-trained residual to a nominal controller (Johannink et al., 2019). Self-compensating VLA compensates at the policy level by adapting the commands sent to the existing controller, without modifying that controller or requiring an explicit dynamics model.

3 Self-compensating VLA

Self-compensating VLA adapts a pretrained VLA to individual robots during deployment to improve its robustness against execution errors. As illustrated in Figure 2, it first obtains a compensation signal from the residual between commanded and executed motion (§3.1), and then uses the signal to update the policy online so that subsequent commands pre-compensate for execution errors (§3.2).

3.1 Compensation Signal

We consider a policy that outputs an action chunk, where each action $a_{t,i}^{\text{policy}}$ is a delta pose, *i.e.*, the commanded change in end-effector pose at step i of chunk t. At each executed step, we measure the

residual by subtracting the commanded delta pose from the executed one:

$$\eta_{t,i}^{\text{obs}} = \Delta x_{t,i} - a_{t,i}^{\text{policy}}, \tag{1}$$

where $x_{t,i}$ is the end-effector pose at step i of chunk t, measured through proprioception, and $\Delta x_{t,i}$ is the achieved change from $x_{t,i}$ to $x_{t,i+1}$. The policy commands a normalized displacement $a_{t,i}^{\text{policy}}$, whereas proprioception measures the executed displacements in physical units. We thus compute the rotational displacement using the SO(3) logarithm map (Murray et al., 2017) and divide the measured displacements by the scale factors used to convert commands to physical units for execution, obtaining $\Delta x_{t,i}$ on the same scale as $a_{t,i}^{\text{policy}}$. Since the residual η^{obs} is computed from issued commands and proprioceptive measurements, it provides a self-supervisory signal for policy adaptation.

3.2 Deployment-time Policy Adaptation

Given the compensation signal, we adapt the policy so that its subsequent commands pre-compensate for the execution errors it reveals. We construct a compensated pseudo-target for this adaptation by subtracting the observed residual from each executed command:

$$a_{t,i}^{\text{target}} = a_{t,i}^{\text{policy}} - \eta_{t,i}^{\text{obs}}. \tag{2}$$

Since the residual $\eta_{t,i}^{\text{obs}}$ is observed only after execution, we cannot use $a_{t,i}^{\text{target}}$ to correct the current chunk but only to update the policy for subsequent chunks. To this end, we apply LoRA (Hu et al., 2022) to the action expert of each base policy (Black et al., 2025b;a), updating only the LoRA parameters ϕ online while keeping the pretrained weights frozen. Since the base policies are flow-matching models (Lipman et al., 2023), we adopt a flow-matching objective that regresses the predicted velocity toward the target velocity defined by the target action chunk:

$$\mathcal{L}_{\text{comp}} = \mathbb{E}_{t,u,\epsilon}\left[\frac{1}{Kd}\left\|v_\theta(s_t, a_t^u, u) - (\epsilon - a_t^{\text{target}})\right\|_2^2\right] + \lambda_{\text{anc}}\|\phi - \phi_0\|_2^2, \tag{3}$$

where K is the chunk length, d is the action dimension, a_t^{target} is the target action chunk, and v_θ is the policy's velocity field. The input s_t comprises the visual observations, proprioceptive state, and language instruction used to generate the original action chunk. The noisy target is $a_t^u = u\epsilon + (1-u)a_t^{\text{target}}$ at flow-matching time $u \in [0,1]$, with $\epsilon \sim \mathcal{N}(0, I)$ and target velocity $\epsilon - a_t^{\text{target}}$. The anchor term, weighted by λ_{anc}, penalizes deviations of the LoRA parameters ϕ from their initialization ϕ_0, limiting parameter drift during online adaptation. We accumulate target chunks from executed action chunks in an online buffer, over which the expectation in Eq. (3) is taken, and update the LoRA parameters during deployment using robot execution feedback.

4 The RoboStress Benchmark

RoboStress is a controlled simulation benchmark for stress-testing VLA robustness to execution errors. It models four joint-level noise components (§4.1), applies each at the corresponding stage of the control pipeline (§4.2), and combines them into seven deployment scenarios (§4.3).

4.1 Joint-level Noise Models

We consider four noise components that cause execution errors: friction, gravity-compensation error, backlash, and compliance. For each, we adopt a joint-level model from the robot dynamics literature (Canudas de Wit et al., 1995; Madsen et al., 2020; Tao & Kokotovic, 1993; Spong, 1987; Gaz et al., 2019) and represent its effect on joint j as a noise term η_j.

Stribeck friction. Joint friction depends nonlinearly on velocity and is elevated near zero speed (Canudas de Wit et al., 1995; Lampaert et al., 2003). We adopt the Coulomb-Stribeck law with a viscous term (Canudas de Wit et al., 1995):

$$\eta_j^{\text{fric}} = -\operatorname{sgn}(\dot{q}_j)\left[F_{c,j} + (F_{s,j} - F_{c,j})\, e^{-(\dot{q}_j/v_{s,j})^2}\right] - \sigma_{v,j}\dot{q}_j, \tag{4}$$

where $\dot{q}_j$ is the joint velocity, *i.e.*, the time derivative of its angle q_j, $\operatorname{sgn}(\cdot)$ is the sign function, $F_{s,j}$ and $F_{c,j}$ are its static and kinetic friction torques, $v_{s,j}$ the Stribeck velocity, and $\sigma_{v,j}$ the viscous

coefficient. We add the resulting torque η_j^{fric} to the joint torque computed by the controller. Since $F_{s,j}$ dominates at low speed $|\dot{q}_j|$, small commands struggle against static friction, degrading low-speed tracking.

Gravity-compensation error. The controller compensates for gravity using estimated link masses, and the mismatch with the true link masses, together with any unknown payload, leaves a configuration-dependent residual torque on each joint (Gaz et al., 2019). We inject this residual as a fraction of the gravitational torque computed by the controller, denoted by η_j^{grav} on joint j:

$$\eta_j^{\text{grav}} = \beta_j \, g_j(\mathbf{q}), \tag{5}$$

where $g_j(\mathbf{q})$ is the gravitational torque computed by the controller at the joint angles $\mathbf{q} = (q_1, \ldots, q_n)$ of the n joints, and $\beta_j \geq 0$ is the residual fraction controlling the strength of the error. η_j^{grav} depends only on the current joint angles $\mathbf{q}$, not on motion history.

Backlash. Gear clearance, which widens with wear (Bittencourt & Axelsson, 2014), creates a deadband on direction reversal in which the motor rotates but the link stays still. We adopt the Tao-Kokotović deadband model (Tao & Kokotovic, 1993), which gives the link angle q_j^l of joint j as a function of its motor angle q_j^m:

$$q_j^l(t) = \begin{cases} q_j^m(t) - B_j & \text{if } q_j^m(t) - q_j^l(t{-}1) > B_j, \\ q_j^m(t) + B_j & \text{if } q_j^m(t) - q_j^l(t{-}1) < -B_j, \\ q_j^l(t{-}1) & \text{otherwise,} \end{cases} \tag{6}$$

where t indexes the time step and B_j is the half-width of the deadband at joint j. We inject backlash by replacing q_j^m with the deflected angle q_j^l from Eq. (6) as the joint position that enters the simulator's integration step (§4.2). The injected noise η_j^{back} represents the angular deviation of the link from the motor due to backlash:

$$\eta_j^{\text{back}} = q_j^l(t) - q_j^m(t), \tag{7}$$

which is bounded by the deadband half-width, $|\eta_j^{\text{back}}| \leq B_j$. η_j^{back} is history-dependent because it depends on the previous link angle $q_j^l(t{-}1)$ as well as the current motor angle $q_j^m(t)$.

Dynamic compliance. A robot joint is not perfectly rigid but compliant, which causes the link to deflect under load, deviating from the commanded angle when the joint torque changes (Spong, 1987; Hardeman, 2008). We take such angular deflection as a type of noise. The deflection on joint j, denoted η_j^{comp}, is formulated as a spring-mass-damper response (Spong, 1987) to the load torque:

$$M_j^{\text{eff}} \, \ddot{\eta}_j^{\text{comp}} + D_j \, \dot{\eta}_j^{\text{comp}} + K_j \, \eta_j^{\text{comp}} = -\tau_j^{\text{load}}, \tag{8}$$

where $\dot{\eta}_j^{\text{comp}}$ and $\ddot{\eta}_j^{\text{comp}}$ are its angular velocity and acceleration, τ_j^{load} is the torque applied at the joint, and M_j^{eff}, D_j, and K_j are the effective inertia, damping, and stiffness of the joint, respectively. The negative sign reflects that the load compresses the transmission, so the link lags behind the motor. The second-order dynamics can produce transient overshoot and oscillation under abrupt load changes. The deflection does not have a fixed value at each step but evolves over time, and we compute it by integrating Eq. (8) online. η_j^{comp} is history-dependent because the deflection reflects the joint's load history, not just the current load.

4.2 Joint-level Noise Injection

As shown in Figure 3, we inject each noise term η_j where it physically arises. The operational space controller (Khatib, 1987) converts the Cartesian action into joint torques τ_j. Friction and gravity-compensation error are added to this torque, and $\tau_j^{\text{load}} = \tau_j + \eta_j^{\text{fric}} + \eta_j^{\text{grav}}$ is applied to the actuator. Backlash and compliance are added to the joint position that enters the simulator's integration step, $q_j^m + \eta_j^{\text{back}} + \eta_j^{\text{comp}}$, with the compliance driven by τ_j^{load} through Eq. (8). The forward dynamics then propagates the perturbed joint torques and positions through the arm, so a perturbation at one joint affects other joints and produces different end-effector errors depending on the arm's pose.

4.3 Deployment Scenarios

A deployed robot rarely exhibits a single noise component of §4.1 in isolation. A robot that is worn, warming up, or carrying a heavy payload is affected by several of these noise components simultaneously, with magnitudes that differ across joints and may change within an episode. RoboStress

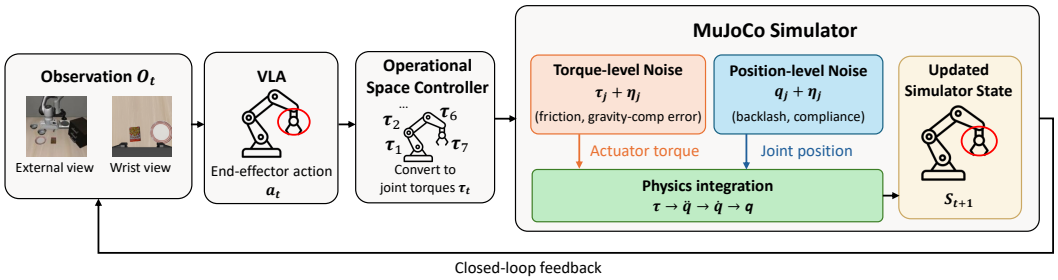

Figure 3: Overview of the joint-level noise injection. The operational space controller (Khatib, 1987) converts end-effector commands into joint torques. Torque-level noise (friction, gravity-compensation error) is added to joint torques, and position-level noise (backlash, compliance) to joint positions before physics integration. The next observation is rendered from the updated state.

Table 1: Deployment scenarios in RoboStress. Checked noise components have increased severity, while unmarked components retain their nominal settings.

Scenario	Physical cause	Fric.	Grav.	Back.	Comp.	Severity over episode	Joints
Heavy Payload	grasped load		✓		✓	While grasping	all
Thermal Drift-Stribeck	temperature change	✓				Linearly increasing	all
Thermal Drift-Backlash	temperature change			✓		Linearly increasing	all
Aged Transmission	gear wear	✓		✓		Fixed	all
Aged Joint-Uniform	gear + bearing wear	✓		✓	✓	Fixed	all
Aged Joint-Shoulder	gear + bearing wear	✓		✓	✓	Fixed	J1–J3
Aged Joint-Elbow	gear + bearing wear	✓		✓	✓	Fixed	J4

therefore combines the four noise components and varies their severity across the seven deployment scenarios of Table 1, which together probe different aspects of a policy's robustness.

Severity factors. Each scenario controls the strength of the four noise components through per-joint severity factors that scale their model parameters in §4.1:

$$(F_{s,j}, F_{c,j}, \sigma_{v,j}) = w_j^f (F_{s,j,0}, F_{c,j,0}, \sigma_{v,j,0}),\ \beta_j = w_j^g \beta_{j,0},\ B_j = w_j^b B_{j,0},\ K_j = K_{j,0}/w_j^c, \quad (9)$$

where the subscript 0 denotes the parameter value before severity scaling, and w_j^f, w_j^g, w_j^b, and $w_j^c \geq 1$ are the severity factors for friction, gravity-compensation error, backlash, and compliance at joint j, respectively. The nominal setting $w = 1$ leaves the model parameters unchanged. As shown in Table 1, each deployment scenario specifies which components have increased severity, which joints are affected, and whether the factors remain fixed or increase linearly over an episode. The factor values for all scenarios are provided in Appendix A.5.

Heavy Payload. *Heavy Payload* models a grasped payload, causing gravity-compensation error and larger joint deflections. We emulate these effects by increasing w_j^g and w_j^c on every joint, while the gripper holds an object, keeping friction and backlash nominal. This scenario therefore tests robustness to execution errors that depend on the arm's pose and load.

Thermal Drift. *Thermal Drift* models execution errors that change with a robot's temperature (Bittencourt & Axelsson, 2014). *Thermal Drift-Stribeck* and *Thermal Drift-Backlash* linearly increase w_j^f and w_j^b, respectively, over the episode. The two variants separately model gradual changes in a velocity-dependent torque component and a history-dependent position component. They test policy adaptation to errors that change within an episode.

Aged Transmission and Aged Joint. Mechanical wear degrades gears and bearings, increasing friction, backlash, and compliance unevenly across joints (Bittencourt & Axelsson, 2014). *Aged Transmission* models gear wear, where worn teeth increase backlash and degraded lubrication raises friction. We therefore increase w_j^f and w_j^b on every joint while keeping compliance nominal. *Aged Joint* additionally models bearing wear by increasing w_j^c, with variants that apply the same severity factors to different sets of joints. *Aged Joint-Uniform* applies these factors to all seven joints, *Aged*

Table 2: Success rates (%) on RoboStress deployment scenarios, averaged over LIBERO tasks. Averages exclude Clean. Best results within each backbone are shown in bold for each row.

	$\pi_{0.5}$				π_0			
Scenario	Base	DR	RobustVLA	Ours	Base	DR	RobustVLA	Ours
Clean	97.0	**97.1**	96.6	95.8	91.1	90.5	92.0	**92.6**
Heavy Payload	44.1	42.1	38.9	**56.8**	42.3	32.0	30.9	**42.8**
Thermal Drift-Stribeck	54.1	54.6	55.9	**60.5**	46.1	46.3	46.0	**50.6**
Thermal Drift-Backlash	83.4	84.5	83.1	**84.8**	75.5	77.3	77.3	**78.8**
Aged Transmission	33.2	39.7	40.2	**41.8**	26.8	22.9	23.8	**28.9**
Aged Joint-Uniform	29.5	36.0	33.8	**38.6**	23.0	22.4	20.0	**24.0**
Aged Joint-Shoulder	43.9	46.4	43.0	**51.2**	34.5	32.9	31.5	**39.2**
Aged Joint-Elbow	55.6	54.0	53.9	**62.3**	43.9	44.5	42.2	**44.9**
Avg.	49.1	51.0	49.8	**56.6**	41.7	39.8	38.8	**44.2**

Joint-Shoulder to joints J1-J3 near the base, and *Aged Joint-Elbow* to joint J4 at the elbow. These three variants test how policy robustness depends on which joints are affected by wear.

5 Experiments

5.1 Experimental Setting

Implementation Details. We adopt π_0 (Black et al., 2025b) and $\pi_{0.5}$ (Black et al., 2025a) as base policies for both the RoboStress and real-world experiments; unless otherwise stated, analyses use π_0. For self-compensating VLA, we attach a rank-4 LoRA (Hu et al., 2022) to each base policy's action expert and optimize it with Adam (Kingma & Ba, 2015) at a learning rate of 10^{-6} and an anchor weight of $\lambda_{\text{anc}} = 10^{-4}$. For real-world experiments, we fine-tune each base policy and RobustVLA on teleoperated demonstrations collected on each Piper arm. On the Piper arms, the policies output absolute joint positions, so we compute residuals and compensated targets in joint space. More implementation details are given in Appendix B.

Evaluation Protocol. We report the success rate averaged over tasks. On RoboStress, which spans all four task suites of LIBERO (Liu et al., 2023) with 10 tasks each, we evaluate the seven deployment scenarios, using 20 episodes per task in each setting; parameter settings of noise components are detailed in Appendix A. For real-world experiments, we evaluate on two AgileX Piper arms with naturally occurring execution errors (§5.3), using 25 episodes per task.

Competitors. We compare against two methods applied to improve VLA robustness, both of which operate at training time. Domain randomization (DR) (Tobin et al., 2017), applied here to the action space, fine-tunes π_0 and $\pi_{0.5}$ under Gaussian noise ($\sigma \sim \mathcal{U}[0, 0.05]$) on the end-effector command, while RobustVLA (Guo et al., 2026) adversarially trains each base policy within an ℓ_∞ ε-ball ($\varepsilon = 0.03$, PGD of Madry et al. (2018)).

5.2 Experiments on RoboStress

Results on deployment scenarios. Table 2 shows that self-compensating VLA outperforms the base policy, DR, and RobustVLA in all seven scenarios with both backbones. It raises average success from 49.1% to 56.6% with $\pi_{0.5}$ and from 41.7% to 44.2% with π_0. In contrast, DR and RobustVLA yield inconsistent gains and fall below the base policy on average with π_0.

Comparing RoboStress Heavy Payload with real payload execution. We replay a pick-and-place demonstration three times on a Piper arm carrying a bowl with 1 kg of weights, and adjust the Heavy Payload parameters to match the resulting end-effector deviations from an unloaded replay. We then replay the task from ten initial layouts in simulation under the adjusted model and under Gaussian action noise at the scale used in RobustVLA's evaluation (Guo et al., 2026), measuring deviations relative to the corresponding unperturbed replays. As shown in Figure 4, the adjusted RoboStress model captures the post-grasp rise in deviation and yields a lower dynamic time warping (DTW) distance to the real mean profile than the Gaussian reference.

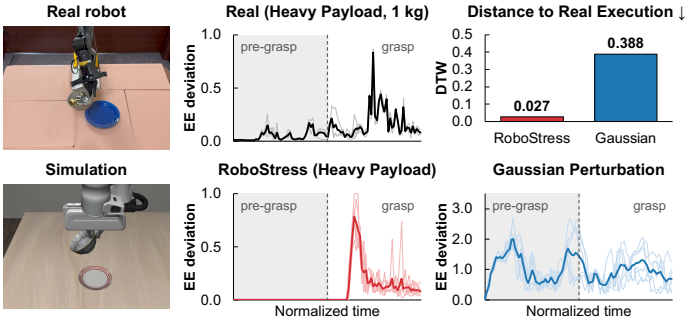

Figure 4: Real-world payload execution deviations compared with RoboStress and Gaussian action noise.

Table 3: Noise analysis.

Noise components	Base	Ours
Stribeck	36.8	**37.8**
Backlash	68.0	**72.1**
Compliance	74.1	**74.2**
Gravity	21.1	**22.0**
Avg.	50.0	**51.5**

Table 4: Method ablation.

	Base	Direct	w/o η^{obs}	Ours
Success (%)	41.7	23.6	40.8	**44.2**

Table 5: Effect of DOB.

	Base	DOB	Ours	DOB+Ours
Success (%)	48	56	**84**	80

Table 6: Real-world success rates (%) on two Piper arms. **Tasks 1–3** involve picking up the gray bowl from next to the plate, on the plastic cabinet, and on the gift box, respectively, and placing it on the plate. **Task 4** involves opening the cabinet's top drawer. Best results are shown in bold.

	Robot A						Robot B					
	$\pi_{0.5}$			π_0			$\pi_{0.5}$			π_0		
Task	Base	RobustVLA	Ours	Base	RobustVLA	Ours	Base	RobustVLA	Ours	Base	RobustVLA	Ours
1	48	52	**84**	44	52	**76**	44	56	**80**	36	56	**60**
2	36	60	**76**	40	28	**80**	36	48	**72**	32	52	**84**
3	48	48	**76**	44	68	**72**	56	52	**84**	52	44	**84**
4	40	48	**68**	32	40	**60**	36	44	**60**	32	44	**56**
Avg.	43	52	**76**	40	47	**72**	43	50	**74**	38	49	**71**

Analysis on noise components. We evaluate each noise component in Table 3. Our method raises the average success rate of π_0 from 50.0% to 51.5%. The smallest gain occurs under compliance, where the base policy already achieves its highest success rate.

Effect of the residual and policy adaptation. We ablate components in our method on π_0 in Table 4. Setting η^{obs}=0 retains online updates but uses the policy's commands as targets. This variant reaches 40.8%, below the base policy and our method. Direct residual correction keeps the policy fixed and subtracts the mean residual of the previous chunk from the next chunk's commands, reaching only 23.6%. Thus, the residual supplies the correction signal, while policy adaptation produces observation-conditioned compensation beyond the fixed chunk-level correction.

5.3 Real-world Experiments

Results on two real robot arms. We evaluate self-compensating VLA on a new AgileX Piper arm (Robot A) and another used for one year (Robot B). The evaluation covers three bowl pick-and-place tasks and one drawer-opening task, as described in Table 6. Even the new arm exhibits command-execution mismatch, with a mean normalized residual of 32.9%, compared with 35.0% on the older arm. As shown in Table 6, self-compensating VLA outperforms the base policies and RobustVLA on every task with both π_0 and $\pi_{0.5}$ on both arms. The representative rollouts in Figure 5 show our method completing both tasks, whereas the base policy fails to grasp the bowl in Task 2 and to open the drawer in Task 4. These results show that self-compensating VLA adapts to naturally occurring execution errors on two physical robots with different usage histories.

Adaptation with external command correction. We evaluate whether our adaptation still helps under external command correction. We add a disturbance observer (DOB) (Chen et al., 2016) that uses joint feedback to estimate execution errors and correct commands before the Piper arm's built-in joint-position controller. As shown in Table 5, on Task 1 of Robot A using $\pi_{0.5}$, DOB alone raises success from 48% to 56%, while adding our adaptation raises it further to 80%, compared with 84% for our adaptation alone. These results indicate that our policy adaptation remains effective alongside external command correction and provides substantial gains beyond DOB alone.

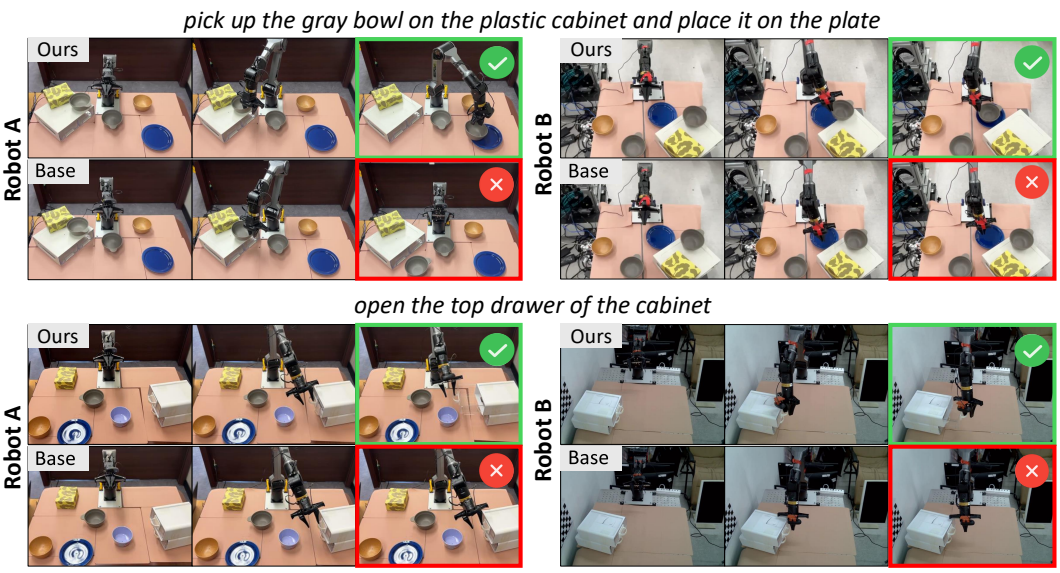

Figure 5: Rollouts of self-compensating VLA (Ours) and the base policy (Base) on Tasks 2 and 4, for Robot A and Robot B. Green and red borders mark success and failure.

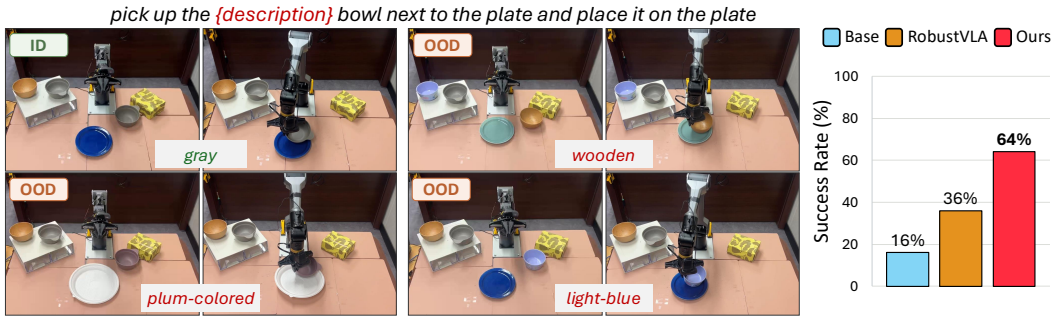

Figure 6: Generalization to unseen objects, evaluated on Robot A. Left: Our rollouts with the gray bowl (ID) and object variants (OOD). Right: Success rates on all OOD variants.

Generalization to objects unseen during fine-tuning. We evaluate Task 1 on Robot A with five object variants absent from the teleoperation fine-tuning data. These variants change the appearance of the bowl or plate and include wooden, plum-colored, and light-blue bowls. We update the instruction to describe each variant and run five episodes per variant, for 25 episodes in total. As shown in Figure 6, self-compensating VLA achieves 64% success with $\pi_{0.5}$, compared with 16% for the base policy and 36% for RobustVLA, showing that its gains extend to unseen object variants.

6 Conclusion

We have studied VLA robustness to execution errors, a challenge for reliable robot deployment. We have proposed self-compensating VLA, which adapts policies online using command-execution residuals to compensate for execution errors without task rewards or labels. We have also introduced RoboStress, a simulation benchmark combining four joint-level error models into seven deployment scenarios. Experiments on RoboStress and two physical Piper arms show improved task success over base policies and training-time robustness methods, with real-world gains extending to objects absent from the fine-tuning demonstrations. These results demonstrate the effectiveness of deployment-time adaptation for improving VLA robustness to robot execution errors. Future work includes incorporating motion estimates from external sensors to reduce reliance on proprioception.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on experimental design, assist with the interpretation of results, and implement methods through code generation and debugging. We have not used generative AI tools to develop the core research ideas or derive the mathematical formulations. We also used them to assist with translation, edit text and captions, and refine the manuscript structure. We reviewed and revised AI-assisted text, checked suggested interpretations against experimental results, and verified AI-assisted code used in the experiments. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work aims to improve the reliability of robot manipulation under execution errors. RoboStress enables controlled stress testing in simulation, reducing the need to expose physical robots to potentially damaging conditions. Our real-world experiments evaluate manipulation tasks on two robot arms. The reported improvements in task success do not constitute safety guarantees; deployment in human environments requires additional safety evaluation and appropriate safeguards.

REPRODUCIBILITY STATEMENT

Section 3 describes the compensation signal, pseudo-target construction, and online adaptation objective. Section 4 and Appendix A detail the RoboStress models, their integration into the simulation pipeline, and the parameters defining the deployment scenarios. Section 5.1 and Appendix B document the evaluation protocol and implementation details, including the action and observation representations, LoRA configuration, online buffer, optimization settings, and computational setup. Appendix D provides additional experimental analyses. We will release the code for self-compensating VLA and RoboStress, including scenario configurations and evaluation scripts, upon publication.

REFERENCES

Zechen Bai, Chen Gao, and Mike Zheng Shou. Evolve-vla: Test-time training from environment feedback for vision-language-action models. *arXiv preprint arXiv:2512.14666*, 2025.

André Carvalho Bittencourt and Patrik Axelsson. Modeling and experiment design for identification of wear in a robot joint under load and temperature uncertainties based on friction data. *IEEE/ASME Transactions on Mechatronics*, 2014.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proc. Conference on Robot Learning (CoRL)*, 2025a.

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 *Proc. Robotics: Science and Systems (RSS)*, 2025b.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl

Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *Proc. Robotics: Science and Systems (RSS)*, 2023.

C. Canudas de Wit, H. Olsson, K.J. Astrom, and P. Lischinsky. A new model for control of systems with friction. *IEEE Transactions on Automatic Control*, 1995.

Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, et al. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

Wen-Hua Chen, Jun Yang, Lei Guo, and Shihua Li. Disturbance-observer-based control and related methods—an overview. *IEEE Transactions on Industrial Electronics*, 63(2):1083–1095, 2016.

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 *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Claudio Gaz and Alessandro De Luca. Payload estimation based on identified coefficients of robot dynamics—with an application to collision detection. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3033–3040. IEEE, 2017.

Claudio Gaz, Marco Cognetti, Alexander Oliva, Paolo Robuffo Giordano, and Alessandro De Luca. Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. *IEEE Robotics and Automation Letters*, 2019.

Hiroaki Gomi and Mitsuo Kawato. Neural network control for a closed-loop system using feedback-error-learning. *Neural Networks*, 6(7):933–946, 1993.

Jianing Guo, Zhenhong Wu, Chang Tu, Yiyao Ma, Xiangqi Kong, Zhiqian Liu, Jiaming Ji, Shuning Zhang, Yuanpei Chen, Kai Chen, Qi Dou, Yaodong Yang, Xianglong Liu, Huijie Zhao, Weifeng Lv, and Simin Li. On robustness of vision-language-action model against multi-modal perturbations. In *Proc. International Conference on Learning Representations (ICLR)*, 2026.

Asher J Hancock, Allen Z Ren, and Anirudha Majumdar. Run-time observation interventions make vision-language-action models more visually robust. In *Proc. International Conference on Robotics and Automation (ICRA)*, 2025.

Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *Proc. International Conference on Learning Representations (ICLR)*, 2021.

Toon Hardeman. *Modelling and identification of industrial robots including drive and joint flexibilities*. PhD thesis, University of Twente, 2008.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proc. International Conference on Learning Representations (ICLR)*, 2022.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *Proc. International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029. IEEE, 2019.

Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 1987.

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 *Proc. Conference on Robot Learning (CoRL)*, 2024.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. International Conference on Learning Representations (ICLR)*, 2015.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: rapid motor adaptation for legged robots. In Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh (eds.), *Proc. Robotics: Science and Systems (RSS)*, 2021.

Vincent Lampaert, Farid Al-Bender, and Jan Swevers. A generalized maxwell-slip friction model appropriate for control purposes. In *2003 IEEE International Workshop on Workload Characterization (IEEE Cat. No. 03EX775)*, volume 4, pp. 1170–1177. IEEE, 2003.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *Proc. International Conference on Learning Representations (ICLR)*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: benchmarking knowledge transfer for lifelong robot learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Proc. Neural Information Processing Systems (NeurIPS)*, 2023.

Changyu Liu, Yiyang Liu, Taowen Wang, Qiao Zhuang, James Chenhao Liang, Wenhao Yang, Renjing Xu, Qifan Wang, Dongfang Liu, and Cheng Han. On-the-fly VLA adaptation via test-time reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40107–40125, 2026.

Hanqing Liu, Shouwei Ruan, Jiahuan Long, Junqi Wu, Jiacheng Hou, Huili Tang, Tingsong Jiang, Weien Zhou, and Wen Yao. Eva-vla: Evaluating vision-language-action models' robustness under real-world physical variations. *arXiv preprint arXiv:2509.18953*, 2025.

Donghai Ma and John M Hollerbach. Identifying mass parameters for gravity compensation and automatic torque sensor calibration. In *Proc. International Conference on Robotics and Automation (ICRA)*. IEEE, 1996.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proc. International Conference on Learning Representations (ICLR)*, 2018.

Emil Madsen, Oluf Skov Rosenlund, David Brandt, and Xuping Zhang. Comprehensive modeling and identification of nonlinear joint dynamics for collaborative industrial robot manipulators. *Control Engineering Practice*, 2020.

Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models. In *Proc. Robotics: Science and Systems (RSS)*, 2025.

Andrea Raviola, Roberto Guida, Andrea De Martin, Stefano Pastorelli, Stefano Mauro, and Massimo Sorli. Effects of temperature and mounting configuration on the dynamic parameters identification of industrial robots. *Robotics*, 2021.

Jean-Jacques E Slotine and Weiping Li. On the adaptive control of robot manipulators. *The International Journal of Robotics Research*, 1987.

M. W. Spong. Modeling and control of elastic joint robots. *Journal of Dynamic Systems, Measurement, and Control*, 1987.

Gang Tao and Petar V. Kokotovic. Adaptive control of systems with backlash. *Automatica*, 1993.

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 *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Hongyin Zhang, Pengxiang Ding, Shangke Lyu, Ying Peng, and Donglin Wang. GEVRM: Goal-expressive video generation model for robust visual manipulation. In *Proc. International Conference on Learning Representations (ICLR)*, 2025a.

Hongyin Zhang, Shuo Zhang, Junxi Jin, Qixin Zeng, Runze Li, and Donglin Wang. RobustVLA: Robustness-aware reinforcement post-training for vision-language-action models. *arXiv preprint arXiv:2511.01331*, 2025b.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proc. Conference on Robot Learning (CoRL)*. PMLR, 2023.

A RoboStress Benchmark Details

A.1 Noise Parameters

Table A1 lists the nominal values of the noise parameters defined in §4.1 for each joint j of the 7-DoF arm. The subscript 0 marks a nominal value, before the severity factors of Eq. (9) are applied. The parameters include those of Stribeck friction ($F_{c,j,0}$, $F_{s,j,0}$, $v_{s,j}$, $\sigma_{v,j,0}$; Eq. (4)), the backlash half-width ($B_{j,0}$; Eq. (6)), those of dynamic compliance (M_j^{eff}, $D_{j,0}$, $K_{j,0}$; Eq. (8)), and the gravity-compensation residual fraction ($\beta_{j,0}$; Eq. (5)). Parameters without the subscript 0, $v_{s,j}$ and M_j^{eff}, are not scaled by the severity factors. Units are given in the table header, and $\beta_{j,0}$ is dimensionless.

The damping is set to $D_{j,0} = 1.4\sqrt{K_{j,0}M_j^{\text{eff}}}$, yielding a damping ratio $\zeta = D_{j,0}/(2\sqrt{K_{j,0}M_j^{\text{eff}}}) = 0.70$ and a natural frequency $f_n = \frac{1}{2\pi}\sqrt{K_{j,0}/M_j^{\text{eff}}}$ of 14.7 Hz (J1–J4) and 22.5 Hz (J5–J7), corresponding to an underdamped second-order response. The backlash half-width $B_{j,0}$ corresponds to approximately $0.05°$ (J1–J4) and $0.075°$ (J5–J7). The residual fraction $\beta_{j,0}$ ranges from 5% to 10%, with the largest values assigned to the shoulder pitch (J2) and elbow (J4).

Joint	$F_{c,j,0}$ (Nm)	$F_{s,j,0}$ (Nm)	$v_{s,j}$ (rad/s)	$\sigma_{v,j,0}$ (Nm·s/rad)	$B_{j,0}$ (rad)	M_j^{eff} (kg·m^2)	$D_{j,0}$ (Nm·s/rad)	$K_{j,0}$ (Nm/rad)	$\beta_{j,0}$
J1	2.175	3.480	0.15	0.348	0.00087	0.70	90.73	6000	0.05
J2	2.175	3.480	0.15	0.348	0.00087	0.70	90.73	6000	0.10
J3	2.175	3.480	0.15	0.348	0.00087	0.70	90.73	6000	0.06
J4	2.175	3.480	0.15	0.348	0.00087	0.70	90.73	6000	0.10
J5	0.480	0.720	0.10	0.048	0.00130	0.20	39.60	4000	0.08
J6	0.480	0.720	0.10	0.048	0.00130	0.20	39.60	4000	0.08
J7	0.480	0.720	0.10	0.048	0.00130	0.20	39.60	4000	0.05

Table A1: Nominal noise parameters per joint j, defined in §4.1.

A.2 Severity Factors

Eq. (9) scales the noise parameters by per-joint severity factors w_j^f, w_j^b, w_j^c, $w_j^g \geq 1$. Table A2 lists the full setting of each factor, namely $w^f = 2$, $w^b = 15$, $w^c = 5$, and $w^g = 4$. For compliance, we also scale the damping as $D_j = D_{j,0}/\sqrt{w_j^c}$ alongside the stiffness scaling $K_j = K_{j,0}/w_j^c$ of Eq. (9) (with M_j^{eff} fixed), so that the damping ratio $\zeta = 0.70$ is preserved as the stiffness drops. The payload factor w_j^g scales the residual fraction $\beta_{j,0}$ and is used in Heavy Payload and in the gravity-compensation setting of Table A3.

Component	Applied to	Full setting
Stribeck friction (w^f)	$F_{c,j}$, $F_{s,j}$, $\sigma_{v,j}$ ($v_{s,j}$ fixed)	$\times 2$
Backlash (w^b)	B_j	$\times 15$
Compliance (w^c)	$K_j = K_{j,0}/w_j^c$, $D_j = D_{j,0}/\sqrt{w_j^c}$ (M_j^{eff} fixed)	$K \div 5$, $D \div \sqrt{5}$
Gravity (w^g)	$\beta_j = w_j^g \beta_{j,0}$	$\times 4$

Table A2: Full setting of each severity factor of Eq. (9). w_j^f, w_j^b, and w_j^c model wear, and w_j^g models payload mismatch.

A.3 Noise Injection Details

The VLA acts at 20 Hz, and the simulator advances each control step through 25 sub-steps of 2 ms. The operational space controller recomputes the joint torques τ_j at every sub-step, and all noise components are recomputed at the same rate, which is the time step t of Eq. (6) and the step of the online integration of Eq. (8). The torque-level components are added to the controller torque before it is written to the actuator, and the sum is saturated at the joint torque limit. The position-level components are implemented as kinematic offsets to the simulator's joint-position state before

each physics integration sub-step. Immediately before each integration sub-step, we add the change in the offset $\eta_j^{\text{back}} + \eta_j^{\text{comp}}$ since the previous sub-step, so that the joint coordinate carries the current offset rather than an accumulated one. The two layers are coupled only through compliance, whose load torque τ_j^{load} in Eq. (8) is the perturbed torque, whereas backlash depends solely on the motor-angle trajectory. No noise is added to the observations; images and proprioception are both obtained from the resulting simulator state. Because the joints are coupled through the mass matrix, a perturbation at one joint also changes the motion of the others. The residual a policy observes is therefore not the injected noise itself but its effect through the arm's dynamics and kinematics at the current configuration. In contrast, perturbations applied directly to end-effector commands, as in prior benchmarks (Zhang et al., 2025b; Guo et al., 2026), do not represent the joint-level sources of these errors.

A.4 Settings of Each Noise Component

Each setting of Table A3 activates a single noise component across all joints while disabling the other three, which is the setting used in Table 3. Table A3 lists the parameter values for each component, separately for joints J1–J4 and J5–J7.

Component	Setting	Injected values (J1–J4 / J5–J7)
Stribeck	friction $\times 2$	$F_c = 4.350/0.960$ Nm, $F_s = 6.960/1.440$ Nm, $\sigma_v = 0.696/0.096$ Nm·s/rad, $v_s = 0.15/0.10$ rad/s
Backlash	$B \times 15$	$B = 0.01305/0.01950$ rad ($0.748^\circ/1.117^\circ$)
Compliance	$K \div 5$, $D \div \sqrt{5}$	$K = 1200/800$ Nm/rad, $D = 40.58/17.71$ Nm·s/rad, $M^{\text{eff}} = 0.70/0.20\,\text{kg·m}^2$ ($\zeta = 0.70$)
Gravity	$\beta \times 4$	$\beta = [0.20, 0.40, 0.24, 0.40, 0.32, 0.32, 0.20]$

Table A3: Settings of noise components. Values are reported as J1–J4 / J5–J7 unless stated otherwise.

A.5 Deployment Scenarios

The deployment scenarios of §4.3 combine the noise components as summarized in Table 1. Table A4 lists the components whose severity is increased in each scenario and their resulting parameter values. The two Thermal Drift scenarios ramp one factor linearly over the episode from the mild setting to the full setting. The Stribeck variant ramps w_j^f from 1.5 to 2 and the Backlash variant ramps w_j^b from 5 to 15. In Heavy Payload, w_j^g and w_j^c take the full setting on every joint while the gripper holds an object and stay nominal otherwise, and $w_j^f = w_j^b = 1$ throughout. The gate turns on when the simulator's grasp check detects a held object and turns off three control steps after release, with a linear ramp-in of 0.1 s. In the three Aged Joint cases, the worn joints are driven to the full setting while the remaining joints stay nominal, with the wear factors applied to all joints (Uniform), to J1–J3 (Shoulder), or to J4 (Elbow). Unless stated otherwise, components not listed in a scenario retain their nominal parameters from Table A1. The Clean condition disables all noise components.

B Implementation Details

B.1 Evaluation Protocol

We evaluate on four LIBERO suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long, and report the success rate averaged over tasks. Every scenario in Tables A3 and A4 is evaluated on all four suites with 20 episodes per task and three seeds, which vary the policy's sampling noise while the initial states are shared, and we report the mean over seeds. For each backbone (Black et al., 2025b;a), all methods start from the same pretrained checkpoint and use the same action horizon ($K = 50$, $n_{\text{exec}} = 5$), per-task initial states, and scenario parameters. The hidden noise states, including the backlash link position and compliance deflection, are reset at the start of

Scenario	Components with increased severity	Injected values
Thermal Drift-Stribeck	Stribeck (ramp)	F_c, F_s, and σ_v ramp $\times 1.5 \rightarrow \times 2.0$ over the episode: $F_c = 3.26 \rightarrow 4.35$ / $0.72 \rightarrow 0.96$ Nm, $F_s = 5.22 \rightarrow 6.96$ / $1.08 \rightarrow 1.44$ Nm, $\sigma_v = 0.522 \rightarrow 0.696$ / $0.072 \rightarrow 0.096$ Nm·s/rad.
Thermal Drift-Backlash	Backlash (ramp)	B ramps $\times 5 \rightarrow \times 15$: $0.00435 \rightarrow 0.01305$ / $0.0065 \rightarrow 0.0195$ rad.
Heavy Payload	Compliance + Gravity (while grasped)	$K = 1200/800$ Nm/rad, $D = 40.58/17.71$ Nm·s/rad, $M^{\text{eff}} = 0.70/0.20\,\text{kg·m}^2$; β_j (J1–J7): 0.20, 0.40, 0.24, 0.40, 0.32, 0.32, 0.20.
Aged Transmission	Stribeck + Backlash	all joints: $F_c = 4.35/0.96$ Nm; $F_s = 6.96/1.44$ Nm; $\sigma_v = 0.696/0.096$ Nm·s/rad; $B = 0.01305/0.01950$ rad.
Aged Joint-Uniform	Stribeck + Backlash + Compliance	all joints: the above Stribeck and backlash parameters plus $K = 1200/800$ Nm/rad, $D = 40.58/17.71$ Nm·s/rad.
Aged Joint-Shoulder	Stribeck + Backlash + Compliance	J1–J3 at the full setting; J4–J7 nominal.
Aged Joint-Elbow	Stribeck + Backlash + Compliance	J4 at the full setting; the other joints nominal.

Table A4: Deployment scenarios. Values are reported as J1–J4 / J5–J7 unless stated otherwise. "Full setting" applies the factors of Table A2, and "nominal" denotes the values of Table A1.

each episode. Because the scenarios evolve in closed loop, their realized execution errors depend on each policy's state trajectory.

B.2 Configuration of Our Method

We freeze each backbone and attach a rank-4 LoRA to its action expert, updating only the LoRA parameters with Adam (learning rate 10^{-6}, anchor weight $\lambda_{\text{anc}} = 10^{-4}$; Eq. (3)). The policy predicts an action chunk of horizon $K = 50$, of which the robot executes the first $n_{\text{exec}} = 5$ steps before the next prediction. After each chunk, we compute the residual of Eq. (1) for the executed steps and store it in a FIFO buffer containing the 32 most recent chunks. Once the buffer is full, we update the LoRA every 8 chunks with one Adam step on a batch of 4 sampled chunks. Since only 5 of the chunk's K=50 predicted steps are executed, only those steps receive residual correction, while the remaining actions retain their original policy values in the target chunk.

In simulation, we carry the LoRA parameters and Adam state across all episodes of the ten tasks in a suite, starting from ϕ_0. We evaluate tasks in LIBERO index order and clear the buffer at each task boundary, so each update uses feedback from the current task. We re-initialize the LoRA parameters, Adam state, and buffer for each suite and scenario.

B.3 Action and Observation Space

The policy outputs a delta action $a = (\Delta p, \Delta\omega, g)$, where $\Delta p \in \mathbb{R}^3$ is the world-frame end-effector translation delta, $\Delta\omega \in \mathbb{R}^3$ the axis-angle rotation delta, and $g \in \{-1, +1\}$ the binary gripper command. In our policy implementation, this 7-dimensional action is represented as an 8-dimensional vector in $[-1, 1]^8$ with one unused channel. The controller maps the normalized action to physical units through $(c_p, c_r) = (0.05\,\text{m},\ 0.5\,\text{rad})$. The proprioceptive observation is $s = (p,\ \text{axisangle}(R),\ g^{\text{qpos}}) \in \mathbb{R}^8$, together with two 224×224 RGB views (third-person and wrist) and the language instruction. To obtain the achieved delta $\Delta x_{t,i}$ of Eq. (1), we use Euclidean subtraction for translation and the $SO(3)$ logarithm map for rotation:

$$\Delta x_{t,i} = \left(\frac{p_{t,i+1} - p_{t,i}}{c_p},\ \frac{\text{Log}_{SO(3)}\left(R_{t,i+1} R_{t,i}^{-1}\right)}{c_r},\ g_{t,i}^{\text{cmd}} \right), \tag{B1}$$

where c_p and c_r rescale the deltas to the normalized action range and the log map is taken along the short path (the rotation of smallest angle). We set the gripper entry to the commanded value g^{cmd},

which excludes the gripper from the residual since its command is binary while its proprioceptive state is continuous, so a residual on it would not be meaningful. Any remaining padding dimensions are retained from the original policy output and receive no residual correction.

B.4 LoRA and Flow-Matching Optimization

The rank-4 LoRA is attached to all linear projections of the 300M-parameter Gemma action expert (query, key, value, and output, and the MLP gate, up, and down projections), giving 2,764,800 trainable parameters, 0.085% of the π_0 backbone's 3.24 B. Following Hu et al. (2022), we initialize $A \sim \mathcal{N}(0, 0.01^2)$ and $B = 0$, so $AB = 0$ initially and the adapted policy matches the base policy before any update. The PaliGemma vision-language backbone is frozen. We minimize the flow-matching loss of Eq. (3) with Adam, drawing one Monte-Carlo (u, ϵ) sample per update with $u \sim \mathrm{Uniform}(0, 1)$ and $\epsilon \sim \mathcal{N}(0, I)$. We set $(\beta_1, \beta_2) = (0.9, 0.999)$, use no weight decay and no learning-rate schedule, and clip gradients to a global ℓ_2-norm of 1.0.

B.5 Computational Setup

All evaluations run on a single NVIDIA A6000 (48 GB) GPU. The codebase builds on JAX 0.5.0 with Flax-NNX, robosuite 1.4.1, and MuJoCo 3.7.0 within the LIBERO benchmark. The control loop runs at 20 Hz (50 ms per step), and MuJoCo integrates the physics at 500 Hz (25 sub-steps of $\Delta t = 2$ ms per control step). Per chunk, multi-step flow-matching inference takes $\sim$80 ms and one LoRA Adam step (batch of 4) takes $\sim$50 ms. The maximum episode length is 220 steps for LIBERO-Spatial, 280 for LIBERO-Object, 300 for LIBERO-Goal, and 520 for LIBERO-Long, corresponding to 11 to 26 s of simulated time.

B.6 Real-Robot Setup

Platform and demonstrations. Each AgileX Piper arm has six revolute joints and a parallel gripper, so the state and action are 7-dimensional. The policy observes two RGB views from base and wrist cameras, the joint state, and the language instruction, and the control loop runs at 30 Hz. On each arm, we collect 30 teleoperated demonstrations for each of the four tasks, recording the commanded absolute joint targets as actions. Each base policy is fine-tuned with the standard flow-matching objective on 50-step action chunks for 20,000 steps, representing joint actions as displacements from the joint state at the start of each chunk.

Real-robot scheduling. The Piper control loop executes $n_{\text{exec}} = 10$ actions per prediction. After each execution window, the client computes the residuals and asynchronously sends feedback to the policy server through a separate WebSocket connection. The server schedules an update every eight feedback chunks. Prediction requests and adaptation updates are handled asynchronously, and completed updates affect subsequent predictions without modifying already generated commands. The server uses the same FIFO buffer, update interval, and optimization settings as in §B.2, unless stated otherwise.

Evaluation state and carry-over. For each arm, task, and backbone, evaluation starts from the corresponding fine-tuned base policy with a zero-effect LoRA initialization, a fresh Adam state, and an empty feedback buffer. We retain the LoRA parameters, Adam state, and feedback buffer across the 25 consecutive evaluation episodes of the same task, and re-initialize all three states at each task boundary. No episodes are reserved solely for adaptation: the reported evaluation episodes also populate the feedback buffer, and online updates begin once the buffer is full.

Residual and pseudo-target. On the physical robots, we compute the residual per joint in radians as

$$\eta_{t,i}^{\text{obs}} = q_{t,i+1} - q_{t,i}^{\text{ref}}, \tag{B2}$$

where $q_{t,i}^{\text{ref}}$ is the joint target predicted by the policy before any controller-side correction, and $q_{t,i+1}$ is the joint state measured at the next control step. The pseudo-target is $a_{t,i}^{\text{target}} = q_{t,i}^{\text{ref}} - \eta_{t,i}^{\text{obs}}$, as in Eq. (2). We exclude the gripper dimension from the residual and retain its original policy target.

The residual and pseudo-target are formed in absolute joint coordinates. Before computing the flow-matching loss, we convert the pseudo-target to the representation used in fine-tuning: we subtract the joint state at the start of the chunk from the six joint coordinates, keep the gripper target unchanged, and apply the checkpoint's action normalization.

DOB implementation. For the DOB baseline, we use an identity nominal model for each joint servo and estimate the equivalent input disturbance with a first-order Q-filter of bandwidth $\omega_Q = 2.0\,\mathrm{s}^{-1}$. At each control step, the DOB clips the estimate z_t to $\hat{d}_t = \mathrm{clip}(z_t, -d_{\max}, d_{\max})$ with $d_{\max} = 0.05\,\mathrm{rad}$ and sends the corrected target $q_t^{\mathrm{DOB}} = q_t^{\mathrm{ref}} - \hat{d}_t$ to the robot controller. We update the estimate at 30 Hz using the elapsed control interval, clamped to $[0.005, 0.2]$ s, and reset the DOB state at the start of each episode. For DOB+Ours, our method computes the residual against the policy target before the DOB correction, $\eta_t^{\mathrm{obs}} = q_{t+1} - q_t^{\mathrm{ref}}$. As the DOB reduces the tracking error, this residual shrinks, so our method adapts only to the error that the DOB does not cancel. Computing the residual against q_t^{DOB} would include the DOB's own correction and could compensate twice.

OOD evaluation. We evaluate OOD generalization using $\pi_{0.5}$. Each OOD evaluation starts from the corresponding fine-tuned $\pi_{0.5}$ policy with the LoRA parameters initialized to ϕ_0, a fresh Adam state, and an empty feedback buffer. We retain the adaptation state across the OOD episodes and object variants.

C Residual Magnitude across Scenarios

Table C5 reports the normalized residual magnitude, $\|\eta^{\mathrm{obs}}\|_2 / \|a^{\mathrm{policy}}\|_2$, measured on π_0 rollouts. For each rollout, we compute the normalized residual over executed non-gripper actions and report the mean across rollouts. Even in the Clean setting, the residual is 23.4%, reflecting the nominal tracking mismatch. Across the seven RoboStress deployment scenarios, it increases to 38.7%–56.3%, or 15.3–32.9 percentage points above Clean, showing that the modeled execution variations induce substantial additional deviations from the policy commands. On the two Piper arms, the residual is 32.9% for Robot A and 35.0% for Robot B, confirming that command-execution mismatch also occurs during physical deployment. The two groups of rows are not on a common scale. The simulation rows use the end-effector residual of Eq. (1) in the normalized action units of §B.3 at 20 Hz with $n_{\mathrm{exec}} = 5$, whereas the real-robot rows use the joint-space residual of §B.6 in radians at 30 Hz with $n_{\mathrm{exec}} = 10$. RoboStress complements these physical experiments by covering execution conditions that are difficult to realize or to vary systematically on physical hardware.

Scenario	Residual (%)
Simulation (RoboStress)	
Clean	23.4
Heavy Payload	48.6
Thermal Drift-Stribeck	47.5
Thermal Drift-Backlash	46.1
Aged Transmission	38.7
Aged Joint-Uniform	48.2
Aged Joint-Shoulder	56.3
Aged Joint-Elbow	47.0
Real robots (AgileX Piper)	
Robot A	32.9
Robot B	35.0

Table C5: Residual magnitude across scenarios, computed as $\|\eta^{\mathrm{obs}}\|_2 / \|a^{\mathrm{policy}}\|_2$ on π_0 rollouts and expressed as a percentage. The simulation rows use the end-effector residual in normalized action units and the real-robot rows use the joint-space residual of §B.6 in radians.

D Additional Results

D.1 Impact of the Anchor Loss

The anchor term in the compensation objective penalizes deviations of the LoRA parameters from their initialization. Table D6 compares success rates with and without this term on π_0, averaged over the four LIBERO suites. Removing the anchor lowers success in all seven deployment scenarios, reducing the average from 44.2% to 41.8% (2.4 percentage points). The largest drop occurs under Heavy Payload (4.6 points), followed by Aged Transmission and Aged Joint-Shoulder (3.7 points each). We therefore retain the anchor term to regularize deployment-time adaptation.

Scenario	w/o anchor	w/ anchor
Heavy Payload	38.2	42.8
Thermal Drift-Stribeck	50.4	50.6
Thermal Drift-Backlash	77.9	78.8
Aged Transmission	25.2	28.9
Aged Joint-Uniform	22.8	24.0
Aged Joint-Shoulder	35.5	39.2
Aged Joint-Elbow	42.8	44.9
Avg.	41.8	44.2

Table D6: Effect of the anchor loss on π_0. Success rates (%) per deployment scenario, averaged over the four LIBERO suites.

D.2 Performance across Severity Levels

To examine how performance changes with the strength of execution errors, we evaluate each deployment scenario at five severity levels, L1 to L5. Each level moves every active component linearly from the *mild* setting (L1) to the full setting of Table A2 (L5), $w_k = w_{L1} + \frac{k-1}{4}(w_{L5} - w_{L1})$ (Table D7). The payload factor w_j^g follows the same rule, and for the two Thermal Drift scenarios the level sets the endpoint of the within-episode ramp while its onset stays fixed, so L1 applies the mild setting throughout the episode. In the Aged Joint scenarios only the worn joints move along the ladder; the remaining joints stay nominal. L5 is the full setting used in Table 2. We use $\pi_{0.5}$ in this analysis.

	Wear factors			Heavy Payload		Thermal Drift ramp end	
Level	Stribeck w^f	Backlash w^b	Compliance $K\div$	$K\div$	$\beta\times$	Stribeck	Backlash
L1 (mild)	1.5	5	2	2	2	1.5→1.5	5→5
L2	1.625	7.5	2.75	2.75	2.5	1.5→1.625	5→7.5
L3	1.75	10	3.5	3.5	3	1.5→1.75	5→10
L4	1.875	12.5	4.25	4.25	3.5	1.5→1.875	5→12.5
L5 (full)	2	15	5	5	4	1.5→2	5→15

Table D7: Severity levels. Each column moves linearly from L1 to L5, and L5 is the setting of Table 2. Compliance divides the stiffness by the listed factor and the damping by its square root ($K_j = K_{j,0}/w_j^c$, $D_j = D_{j,0}/\sqrt{w_j^c}$), which keeps $\zeta = 0.70$ at every level. Wear factors apply to the worn joints of each scenario, that is, all joints for Aged Transmission and Aged Joint-Uniform, J1–J3 for Shoulder, and J4 for Elbow. Thermal Drift entries give the ramp as onset→end multiplier.

Figure D1 shows the results. Success generally decreases as severity increases for both methods, and the rate of decrease depends on the scenario. Heavy Payload is the most sensitive as the success rate of self-compensating VLA averaged over the four suites drops from 84.3% at L1 to 56.8% at L5. Thermal Drift-Stribeck and Thermal Drift-Backlash are the least sensitive, dropping from 65.2% to 60.5% and from 92.4% to 84.8%, respectively, and success on LIBERO-Spatial under

Aged Joint-Elbow remains around 90% at every level. Across suites, LIBERO-Long is the most affected, falling to about two-thirds of its L1 success rate, since errors accumulate over its longer horizon. Self-compensating VLA generally matches or outperforms the base policy at nearly every severity level. The gap widens with severity on Heavy Payload and the Aged Joint scenarios, most notably on LIBERO-Object and LIBERO-Long, while it stays roughly constant on the Thermal Drift scenarios.

D.3 Qualitative Examples of the RoboStress Benchmark

We provide videos that show the RoboStress benchmark in the supplementary video (`supplementary_video.mp4`). The clips contrast the Clean reference, which injects no noise, with the deployment scenarios of §4.3, showing how the injected noise components of each scenario visibly perturb the robot's execution relative to the clean case.

D.4 Qualitative Results on Real Robots

The same video (`supplementary_video.mp4`) further presents real-robot rollouts on the two AgileX Piper arms, Robot A and Robot B, across the four tasks (Tasks 1-4). For each task and robot, we show the third-person camera view, comparing the base policy (Black et al., 2025b;a) with self-compensating VLA. The clips show that the base policy often fails to grasp the bowl or the drawer handle under the arm's execution errors, whereas self-compensating VLA shows successful rollouts on both robots.

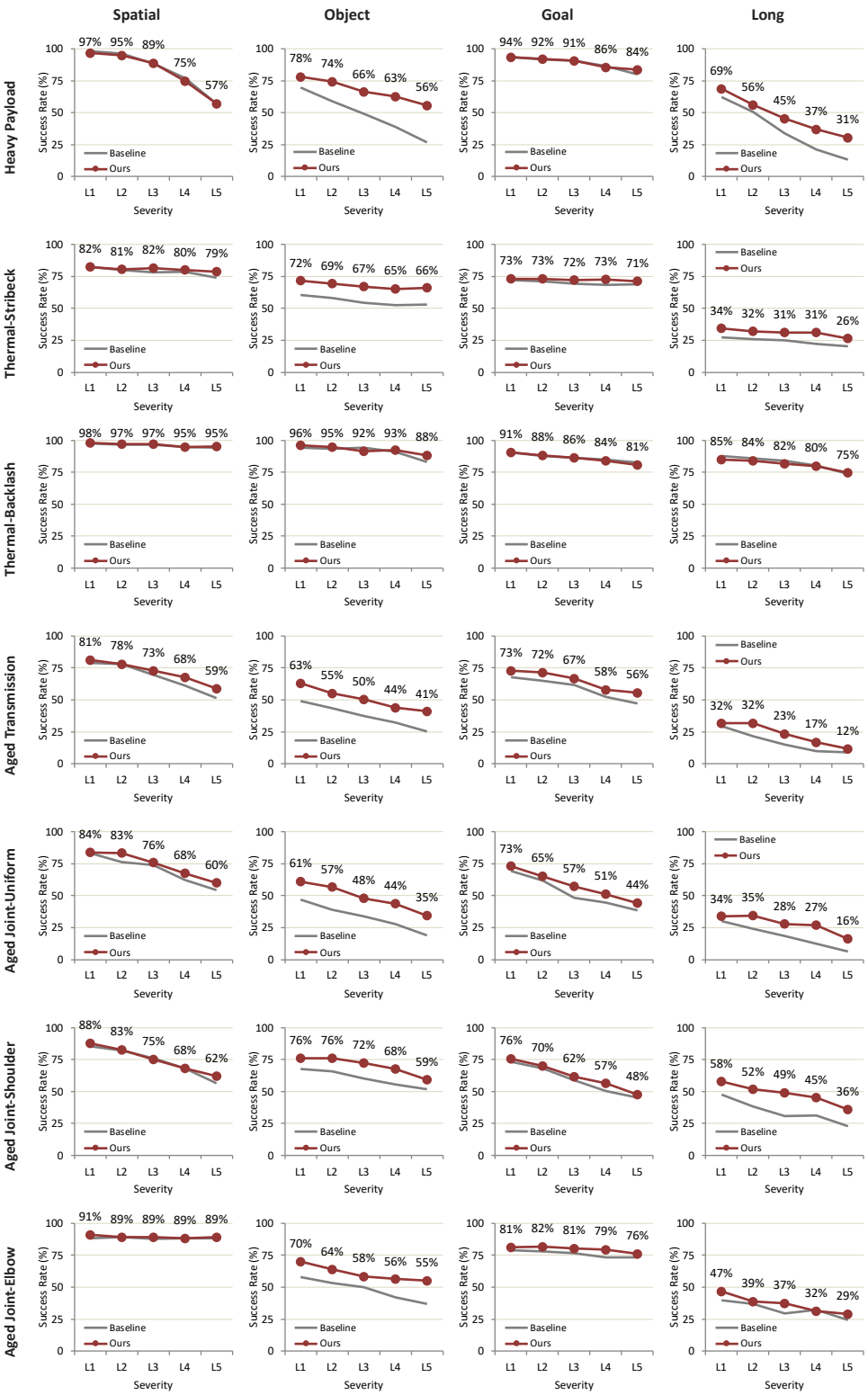

Figure D1: Success rate (%) of the base policy and self-compensating VLA with $\pi_{0.5}$ across five severity levels (L1 to L5) for each deployment scenario (rows) and LIBERO suite (columns) in RoboStress, averaged over the same three seeds as Table 2.