FLOWSHIFT: RISK-AWARE TEST-TIME SCALING FOR VISION-LANGUAGE-ACTION MODELS WITH VERIFIER-DRIVEN GUIDANCE AND SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models often degrade under test-time distribution shifts, when training and deployment distributions diverge. Existing test-time scaling methods mainly select among completed action candidates, leaving the generation process constrained by pretrained flow dynamics and the verifier fixed under deployment changes. To address these limitations, we propose **FlowShift**, a plug-and-play, risk-aware test-time scaling framework that adaptively shifts the action distribution while keeping the VLA backbone frozen. (1) FlowShift first introduces a **Test-Time Adaptive Task-Progress Verifier**, which adapts latent prompts through self-supervised state prediction to remain responsive to deployment changes. The adapted verifier provides two complementary signals for subsequent action generation: reward gradients for guidance and scalar rewards for evaluation. (2) Building on these signals, **Verifier-Guided Stochastic Flow Sampling** converts selected ODE updates into verifier-guided SDE transitions, using reward gradients to steer intermediate states toward higher-reward regions and scalar rewards to select promising particles. This process progressively shifts the action distribution toward a reward-aligned target. (3) To prevent excessive distribution shifts, FlowShift further introduces a **Risk-Aware Router** that models multi-timestep flow features and trajectory history to estimate execution risk. It activates stochastic flow sampling for high-risk action chunks while retaining the original ODE path for low-risk ones, thereby controlling distribution shift. FlowShift improves $\pi_{0.5}$ from 83.9% to 85.0% on LIBERO-Plus, from 63.9% to 68.9% and 39.0% to 46.2% on RoboTwin 2.0 under the *Easy* and *Hard* settings, respectively, and from 56.7% to 69.2% in real-world manipulation.

1 INTRODUCTION

Recent advances in Vision-Language-Action (VLA) models have demonstrated strong robotic manipulation capabilities through large-scale vision-language pretraining and robot demonstrations (Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025). Despite this progress, deploying VLA models beyond their training distributions remains challenging. Changes in object appearance, camera viewpoint, illumination, or scene layout can substantially degrade performance (Fei et al., 2025; Li et al., 2026), as the deployment environment may deviate from the training distribution. This is especially challenging in the real world, where covering all variations is infeasible. Although additional fine-tuning can adapt a policy to a target domain, it requires new demonstrations and model updates across deployment conditions. This motivates test-time approaches that adapt action generation to distribution shifts without modifying the pretrained VLA backbone.

Test-time scaling (TTS) provides a promising alternative by allocating additional inference computation to improve action predictions. Existing approaches commonly generate multiple complete action candidates and select them using a verifier or reward model (Kwok et al., 2025; Jang et al., 2026; Nakamoto et al., 2024). While effective, such post-hoc selection has two fundamental limitations. First, verifier feedback only selects among completed candidates without altering the generation process. The search therefore remains constrained by the original policy distribution, which may be misaligned with the deployment environment. Second, the verifier is typically fixed after training and cannot adapt its judgment to deployment changes. As illustrated in Fig. 1, these limita-

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.

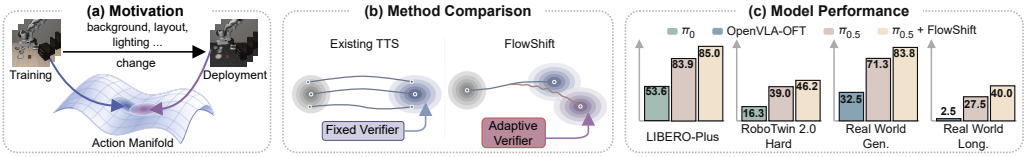

Figure 1: (a) **Motivation:** Deployment changes can shift VLA action predictions away from the training distribution. (b) **Method Comparison:** Unlike existing TTS with post-hoc selection and a fixed verifier, FlowShift adaptively guides intermediate flow refinement. (c) **Performance:** FlowShift improves VLA performance across simulation and real-world tasks.

tions motivate integrating verification directly into action generation, allowing test-time feedback to reshape the action distribution rather than merely select from existing candidates.

To address these limitations, we propose **FlowShift**, a plug-and-play, risk-aware test-time scaling framework that adaptively shifts the action distribution while keeping the VLA backbone frozen. Instead of using verifier feedback only to evaluate completed action chunks, FlowShift integrates verification into flow-matching generation, guiding intermediate states toward higher-reward regions and shifting the action distribution toward a reward-aligned target. However, repeated flow refinement may excessively deviate from the original policy. FlowShift therefore combines verifier-guided refinement with risk-aware routing to regulate the timing and extent of distribution shifting.

The process begins with a **Test-Time Adaptive Task-Progress Verifier** that provides deployment-responsive guidance for action generation. Given the current observation, language instruction, and candidate action chunk, the verifier estimates a task-progress reward. At test time, FlowShift adapts latent verifier prompts through self-supervised state prediction while keeping both the VLA and verifier backbones frozen. This keeps verification responsive to deployment changes without fine-tuning and provides reward gradients for guidance and scalar rewards for evaluation.

Building on these adaptive reward signals, **Verifier-Guided Stochastic Flow Sampling** intervenes in action generation, as illustrated in Figure 1(b). At selected timesteps, FlowShift converts deterministic ordinary differential equation (ODE) updates into stochastic differential equation (SDE) transitions to explore alternative intermediate states. Reward gradients guide these transitions toward higher-reward regions, while scalar rewards evaluate and select promising particles. Repeated guidance and selection progressively redirect the flow toward a reward-aligned action distribution.

Since such iterative refinement can accumulate distributional deviation when applied indiscriminately, FlowShift further introduces a **Risk-Aware Router** to regulate the intervention. The router jointly models the evolution of intermediate features across multiple flow-matching timesteps and the temporal history of executed action chunks to estimate execution risk. High-risk chunks activate verifier-guided stochastic flow sampling, whereas low-risk chunks retain the original ODE generation path. In this way, the router controls when and to what extent the action distribution is shifted, preventing excessive distribution deviation over the execution trajectory.

We evaluate FlowShift extensively on simulation benchmarks and real-world robotic manipulation. On LIBERO-Plus, FlowShift improves the success rate of $\pi_{0.5}$ from 83.9% to 85.0%. On RoboTwin 2.0, FlowShift improves the success rate of $\pi_{0.5}$ from 63.9% to 68.9% under the *Easy* setting. Under the more challenging *Hard* setting, the success rate increases from 39.0% to 46.2%. In the real world, we evaluate FlowShift on a 14-DoF bimanual robot across six challenging manipulation tasks, where it improves the overall success rate of $\pi_{0.5}$ from 56.7% to 69.2%.

In summary, our contributions are as follows:

- We introduce a **Test-Time Adaptive Task-Progress Verifier** and **Verifier-Guided Stochastic Flow Sampling**, converting selected ODE updates into verifier-guided SDE transitions that leverage adaptive reward signals for gradient guidance and particle selection during flow matching.
- We develop a **Risk-Aware Router** that models multi-timestep flow evolution and trajectory history to estimate execution risk, selectively activating stochastic flow refinement while retaining the original ODE path for low-risk action chunks to prevent excessive distribution shifts.
- We develop **FlowShift**, a plug-and-play, risk-aware test-time scaling framework that adaptively shifts the action distribution under deployment changes while keeping the pretrained VLA backbone frozen, with consistent gains in simulation and real-world tasks.

2 Related Work

Vision-Language-Action Models. Vision-Language-Action (VLA) models map visual observations and language instructions to executable robot actions. Early VLAs formulate action generation as autoregressive token prediction (Brohan et al., 2023; Kim et al., 2024), while recent approaches adopt continuous action generation and dual-system architectures that pair a large vision-language backbone with a lightweight action expert (Black et al., 2024; Intelligence et al., 2025). Despite strong performance in training environments, these policies learn fixed conditional action distributions from offline demonstrations. Deployment shifts in visual appearance, object configuration, robot state, or task specification can therefore bias predictions toward training-distribution modes and degrade out-of-distribution performance (Fei et al., 2025; Li et al., 2026).

Test-Time Scaling for VLA Models. Existing VLA test-time scaling methods, including RoboMonkey, MG-Select, and TACO, generate multiple action candidates and select them using verifiers or confidence signals (Kwok et al., 2025; Jang et al., 2026; Yang et al., 2025). TapSampling performs sampling and verification in a learned action latent space with a task-progress verifier (Zhao et al., 2026). VLS instead steers diffusion or flow-matching trajectories using VLM-synthesized differentiable geometric rewards (Liu et al., 2026). Despite this progress, reshaping intermediate trajectories toward deployment-compatible action distributions without post-hoc selection or explicit reward construction remains challenging. We address this with Verifier-Guided Stochastic Flow Sampling and a Test-Time Adaptive Verifier for guidance and selection.

Test-Time Training for VLA Models. Test-time training adapts VLA models using self-supervised objectives on unlabeled deployment interactions. TTT-VLA jointly trains latent prompts and the action-generation model with a state-grounding objective, then updates only the prompts online (Zhang et al., 2026a). T^3VF adapts a visual-foresight pathway by matching predicted future observations to subsequent observations (Park et al., 2026), while RoboTTT inserts fast-weight layers and requires dedicated long-context training (Jiang et al., 2026). Despite updating few parameters at test time, these methods remain coupled to the VLA through modified action-generation or prediction architectures. In contrast, we keep the VLA backbone unchanged and adapt only latent prompts in a lightweight task-progress verifier via self-supervised state prediction.

3 Method

3.1 Problem Formulation and Framework Overview

Problem Formulation. We consider closed-loop manipulation with a pretrained flow-based VLA. At step t, observation O_t, proprioceptive state $\mathbf{s}_t$, and instruction ℓ condition an action chunk $\mathbf{A}_t \in \mathbb{R}^{H\times D}$, with $\mathbf{c}_t = \Phi_{\mathrm{VLM}}(O_t, \mathbf{s}_t, \ell)$. At inference, $\mathbf{A}_t^0 = \epsilon,\ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\frac{d\mathbf{A}_t^\tau}{d\tau} = \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t)$. Forward Euler gives $\mathbf{A}_t^{\tau+\delta} = \mathbf{A}_t^\tau + \delta \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t)$, forming a deterministic ODE path. Under distribution shifts, this fixed flow may remain biased toward training modes; we instead guide intermediate states toward higher-success regions with the VLA frozen.

Framework Overview. As illustrated in Fig. 2, FlowShift consists of three components: a Risk-Aware Router, a Test-Time Adaptive Task-Progress Verifier, and Verifier-Guided Stochastic Flow Sampling. Together, they enable risk-conditioned flow refinement with adaptive verification while keeping the VLA frozen. We detail each component in the following sections.

3.2 Test-Time Adaptive Task-Progress Verifier

A verifier trained only on offline demonstrations may become unreliable under deployment shifts. To provide deployment-responsive guidance, we introduce a *test-time adaptive task-progress verifier* that combines task-progress estimation with self-supervised test-time adaptation. Figure 3 illustrates its architecture, with detailed implementation and training settings provided in Appendix C.1.

Task-Progress Data Construction. We derive task-progress supervision directly from the temporal structure of successful expert demonstrations, without additional manual annotation. A trajectory of T decision steps is represented as $\tau = \{\ell, (O_i, \mathbf{s}_i, \mathbf{A}_i)\}_{i=1}^T$. Assuming task progress increases linearly along a successful trajectory, we assign each step a normalized progress value $p_i = i/T$.

Given an observation O_i, the subsequent k expert action chunks $\mathbf{A}_{i:i+k-1}$ form a positive sample with target $y_i^+ = p_{i+k} - p_i = k/T > 0$. For negative supervision, we reverse the preceding

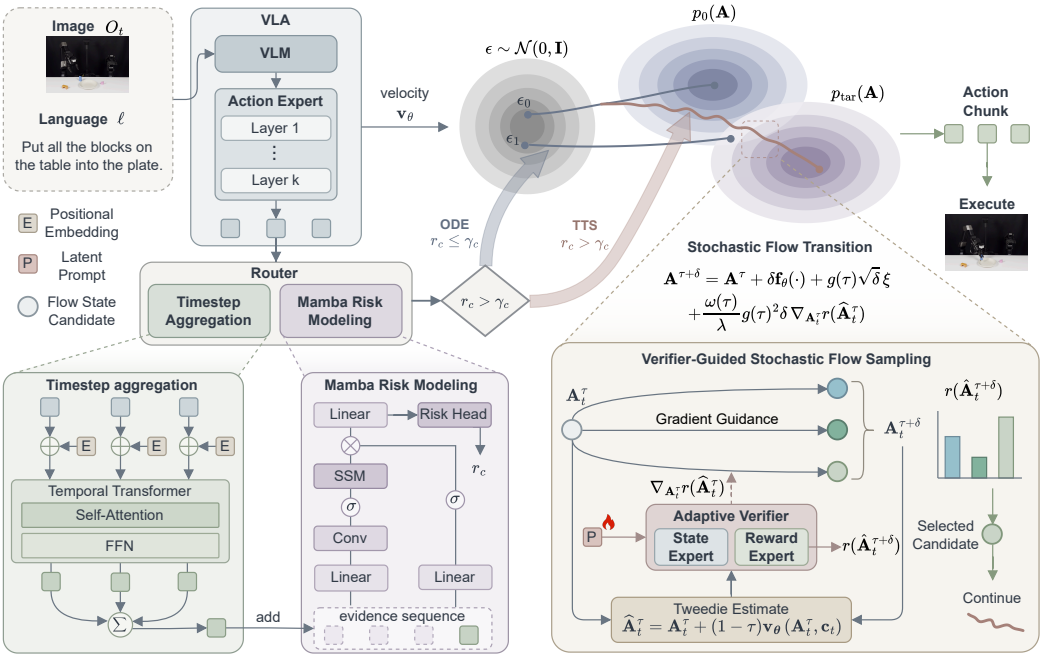

Figure 2: Overview of FlowShift. The risk-aware router estimates execution risk from early flow features and cross-chunk history, retaining ODE inference for low-risk chunks and activating TTS for high-risk ones. During TTS, stochastic transitions explore intermediate states, while the adaptive verifier provides reward gradients for guidance and scalar rewards for selection, shifting the action distribution toward $p_{\text{tar}}(\mathbf{A})$. Only the verifier's latent prompts are adapted during deployment.

k expert action chunks to obtain $\mathbf{A}^r_{i:i-k+1}$, with target $y_i^- = p_{i-k} - p_i = -k/T < 0$. Thus, successful demonstrations alone provide both advancement and regression supervision.

State-Predictive Verifier Architecture. Given observation O_t, instruction ℓ, and action chunk $\mathbf{A}_t \in \mathbb{R}^{H \times D}$, the verifier predicts a scalar reward $r_{\boldsymbol{\phi}}(O_t, \ell, \mathbf{A}_t; \mathbf{P})$, where larger values indicate greater task progress. For test-time adaptation, we introduce learnable latent prompts $\mathbf{P}$ and a State Expert that predicts the proprioceptive state $\mathbf{s}_t$ as a self-supervised signal.

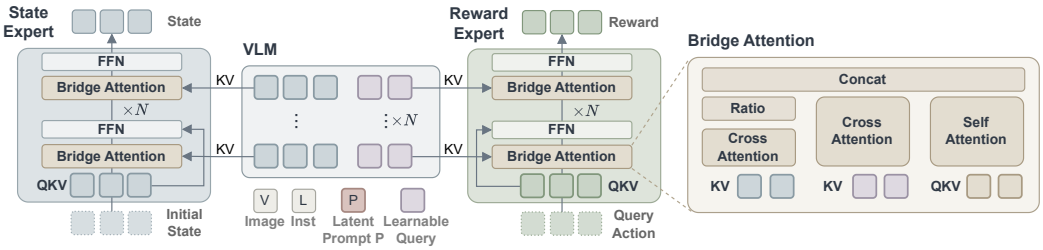

Figure 3: Architecture of the test-time adaptive task-progress verifier. The Prismatic VLM encodes visual observations and language instructions together with learnable Query and latent prompts. The Reward Expert evaluates candidate action chunks, while the State Expert provides the self-supervised signal for adapting the latent prompts at test time.

Built upon VLA-Adapter (Wang et al., 2025), the verifier uses a Prismatic VLM with Qwen2.5-0.5B (Yang et al., 2024) to encode two RGB views and the instruction. Learnable Query tokens extract action-relevant features, while Bridge Attention fuses VLM features with candidate actions or proprioceptive states for the two experts. Latent prompts condition the shared VLM features and serve as the test-time adaptation interface. The Reward Expert predicts task progress, whereas the State Expert predicts $\mathbf{s}_t$. State-predictive latent prompts adapt the verifier to deployment shifts while keeping the VLA frozen. Only the latent prompts are updated at test time; see Appendix C.1.1.

Offline Fine-Tuning. The verifier follows separate offline and test-time optimization paths. During offline fine-tuning, we formulate task-progress prediction as regression and optimize the Reward

Expert using the constructed progress targets with an L1 loss:

$$\mathcal{L}_{\text{reward}} = \|r_{\boldsymbol{\phi}}(O_t, \ell, \mathbf{A}_t; \mathbf{P}) - y_t\|_1 , \tag{1}$$

where y_t is the progress target. The State Expert simultaneously predicts $\mathbf{s}_t$ using $\mathcal{L}_{\text{state}} = \|\widehat{\mathbf{s}}_t - \mathbf{s}_t\|_1$. The two losses follow separate gradient routes: $\mathcal{L}_{\text{reward}}$ updates the Reward Expert, ActionQuery tokens, and parameter-efficient VLM adapters, whereas $\mathcal{L}_{\text{state}}$ updates only the State Expert and latent prompts $\mathbf{P}$, preventing the prompts from directly fitting task-progress labels. Further fine-tuning details are provided in Appendix C.1.2.

Test-Time Adaptation. During deployment, only $\mathbf{P}$ is adapted using observed proprioceptive states, while the VLM and both experts remain frozen. Each generated action chunk adds one state-prediction sample to a fixed-capacity buffer $\mathcal{B}$. Once the buffer is full, we optimize

$$\mathcal{L}_{\text{TTA}} = \frac{1}{|\mathcal{B}|} \sum_{j \in \mathcal{B}} \|\widehat{\mathbf{s}}_j(\mathbf{P}) - \mathbf{s}_j\|_1 . \tag{2}$$

We perform K updates on the buffered samples, modifying only the latent prompts as $\mathbf{P} \leftarrow \mathbf{P} - \eta \nabla_{\mathbf{P}} \mathcal{L}_{\text{TTA}}$, where η is the test-time learning rate. The buffer is cleared after adaptation, while the prompts are retained and accumulate updates within each episode; both are reset at the next episode. Further adaptation details are provided in Appendix C.1.3.

3.3 Verifier-Guided Stochastic Flow Sampling

A deterministic flow-based VLA generates each action chunk along a single ODE trajectory, limiting recovery once the trajectory enters an unfavorable region. We therefore introduce verifier-guided stochastic flow sampling to explore alternative intermediate states and steer them toward higher-reward regions without modifying the pretrained VLA.

For brevity, we denote the pretrained action distribution as $p_0(\mathbf{A}) = p(\mathbf{A} \mid \mathbf{c}_t; \boldsymbol{\theta})$ and the verifier reward as $r(\mathbf{A}) = r_{\boldsymbol{\phi}}(O_t, \ell, \mathbf{A}; \mathbf{P})$. Reward-guided steering seeks a higher-reward distribution while remaining close to the pretrained policy. We therefore define $p_{\text{tar}} = \arg\max_q [\mathbb{E}_{\mathbf{A}\sim q}[r(\mathbf{A})] - \lambda D_{\text{KL}}(q \| p_0)]$, whose solution is

$$p_{\text{tar}}(\mathbf{A}) = \frac{1}{Z} p_0(\mathbf{A}) \exp\left(\frac{r(\mathbf{A})}{\lambda}\right), \tag{3}$$

where Z is the normalization constant and $\lambda > 0$ controls the trade-off between reward improvement and deviation from the pretrained action distribution. The derivation is provided in Appendix A.1. Since direct sampling from p_{tar} is intractable, we approximate this target through verifier-guided gradient steering and particle search over stochastic flow transitions.

Stochastic Flow Transition. At predefined stochastic steps, we convert the deterministic ODE dynamics in Sec. 3.1 into a same-marginal SDE. For a linear flow path, the score and SDE drift are

$$\mathbf{s}_{\boldsymbol{\theta}} = \frac{\tau \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t) - \mathbf{A}_t^\tau}{1 - \tau}, \qquad \mathbf{f}_{\boldsymbol{\theta}} = \mathbf{v}_{\boldsymbol{\theta}} + \frac{1}{2} g(\tau)^2 \mathbf{s}_{\boldsymbol{\theta}}, \tag{4}$$

where $g(\tau)$ controls the diffusion strength. A detailed derivation is provided in Appendix A.2.

At each selected step, the resulting SDE is discretized as

$$\mathbf{A}_t^{\tau+\delta} = \mathbf{A}_t^\tau + \delta\, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t) + g(\tau)\sqrt{\delta}\, \xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{5}$$

The discretization is detailed in Appendix A.3. The Gaussian perturbation produces local proposals from the same flow state for verifier-driven guidance and selection.

Verifier-Driven Gradient Guidance and Particle Search. The task-progress verifier guides stochastic sampling in two complementary ways: its reward gradient steers proposal generation, while its scalar reward evaluates and selects the resulting particles. Given the current flow state $\mathbf{A}_t^\tau$, we estimate its clean endpoint using Tweedie's formula, $\widehat{\mathbf{A}}_t^\tau = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t)$. According to the reward-aligned distribution in Eq. equation 3, the reward-induced guidance at the current flow state is approximated by $\frac{1}{\lambda} \nabla_{\mathbf{A}_t^\tau} r(\widehat{\mathbf{A}}_t^\tau)$. We incorporate this guidance directly into the stochastic transition, yielding the i-th proposal

$$\mathbf{A}_{t,i}^{\tau+\delta} = \mathbf{A}_t^\tau + \delta\, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t) + g(\tau)\sqrt{\delta}\, \xi_i + \frac{\omega(\tau)}{\lambda} g(\tau)^2 \delta\, \nabla_{\mathbf{A}_t^\tau} r(\widehat{\mathbf{A}}_t^\tau), \tag{6}$$

where $\xi_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\omega(\tau)$ controls the guidance strength.

Each proposal $\mathbf{A}_{t,i}^{\tau+\delta}$ is converted to a clean-action estimate via the Tweedie estimator and scored by the verifier, while tracking the best reward $r^\star$. If a proposal improves $r^\star$, it is accepted and the unused number of function evaluations (NFE) budget rolls over to the next stochastic step under Rollover Budget Forcing (RBF) (Kim et al., 2025a). Otherwise, the top-scoring proposal is retained.

3.4 Risk-Aware TTS Routing via Multi-Scale Temporal Modeling

Verifier-guided TTS shifts the pretrained action distribution, so the degree of distribution shift must be controlled. We define execution risk as the likelihood that the current ODE-generated action chunk fails and use it to decide TTS activation, thereby regulating the shift degree. A lightweight router predicts this risk from within-chunk flow evolution and cross-chunk trajectory history. High-risk chunks activate TTS, while low-risk chunks retain ODE inference.

Risk-Aware TTS Routing. TTS should be applied selectively because refinement may improve a suboptimal ODE prediction but also perturb an already reliable one. Let $V(\mathbf{A})$ be the task value and $\mathbf{d}_c$ the TTS-induced displacement for chunk c. If V is locally L-smooth,

$$\Delta V_c \geq \nabla_{\mathbf{A}} V(\mathbf{A})^\top \mathbf{d}_c - \frac{L}{2}\|\mathbf{d}_c\|_2^2. \tag{7}$$

The second term penalizes excessive deviation from the original action. If $D_c = D_{\mathrm{KL}}(p_c^{\mathrm{TTS}} \| p_c^{\mathrm{ODE}}) \leq D_{\max}$, then $D_{\mathrm{KL}}(P_g \| P_{\mathrm{ODE}}) \leq D_{\max}\mathbb{E}_{P_g}[N_{\mathrm{TTS}}]$, so limiting TTS activation bounds cumulative shift. Accordingly, TTS is activated only when the predicted failure risk r_c exceeds γ_c; otherwise, ODE inference is retained. See Appendix B for derivations.

Flow-Timestep Evidence Aggregation. To assess execution risk before stochastic steering, we aggregate action features across early flow-matching steps as risk evidence.

For action chunk c, let $\mathbf{F}_{c,m} \in \mathbb{R}^{H \times d}$ denote the final-layer action-head features at the m-th selected flow step, where H is the action horizon and d the hidden dimension. We first pool the action tokens as $\mathbf{z}_{c,m} = \frac{1}{H}\sum_{h=1}^{H} \mathbf{F}_{c,m,h}$. The pooled features from M selected steps are processed by timestep-specific normalization and projection, learned temporal embeddings, and a lightweight temporal Transformer with weighted pooling. We denote this aggregation as $\mathbf{e}_c = \mathrm{Agg}(\mathbf{z}_{c,1}, \ldots, \mathbf{z}_{c,M})$. The resulting $\mathbf{e}_c$ summarizes the evolution of the current action representation along the early ODE trajectory and serves as evidence for risk prediction.

Mamba Risk Modeling. Chunk-level flow evidence alone cannot capture prior closed-loop execution. We therefore model $\mathbf{e}_{1:c}$ with a causal Mamba backbone (Gu & Dao, 2023), yielding $\mathbf{h}_c = \mathrm{Mamba}(\mathbf{e}_{1:c})$ and $r_c = \sigma(\mathbf{w}_r^\top \mathbf{h}_c + b_r)$. Its selective state update retains useful history and suppresses irrelevant cues, yielding trajectory-aware risk features.

The router is trained with rollout-level success and failure labels. Following SAFE (Gu et al., 2026), we calibrate $\gamma_c = \mu_c + \kappa_c$ from successful held-out rollouts at coverage $1 - \alpha_{\mathrm{cp}}$. At inference, early-step features are aggregated into $\mathbf{e}_c$ and combined with history to predict r_c. If $r_c \leq \gamma_c$, ODE inference is retained; otherwise, TTS is activated.

4 Experiments

We evaluate FlowShift on simulation benchmarks and real-world manipulation, focusing on robust performance under diverse test-time distribution shifts and deployment conditions.

4.1 Evaluation on Simulation Benchmarks

Simulation Benchmarks. To evaluate FlowShift under diverse test-time distribution shifts, we conduct experiments on LIBERO-Plus (Fei et al., 2025) and RoboTwin 2.0 (Chen et al., 2025). LIBERO-Plus extends LIBERO (Liu et al., 2023) with perturbations in camera, language, appearance, and scene layout; we report category-wise and overall success rates. On RoboTwin 2.0, we evaluate all 50 bimanual tasks under *Easy* and *Hard* settings over 100 rollouts per task.

Implementation Details. FlowShift is built on the frozen $\pi_{0.5}$ backbone (Intelligence et al., 2025). The task-progress verifier is fine-tuned for 30k steps with a global batch size of 32. The router aggregates action-head features from flow steps $\{1, 3, 5, 7\}$ using a single-layer Mamba and is trained

Table 1: Results on LIBERO-Plus under the zero-transfer setting. We report success rates (%) across seven perturbation categories and overall performance. † denotes results reproduced by us. Best results are highlighted in **bold**, with FlowShift gains shown in green.

Method	Camera	Robot	Language	Light	Background	Noise	Layout	Total
OpenVLA-OFT (Kim et al., 2025b)	56.4	31.9	79.5	88.7	93.3	75.8	74.2	69.6
π_0 (Black et al., 2024)	13.8	6.0	58.8	85.0	81.4	79.0	68.9	53.6
π_0-Fast (Pertsch et al., 2025)	65.1	21.6	61.0	73.2	73.2	74.4	68.8	61.6
VLA-Adapter (Wang et al., 2025)	36.4	37.4	73.8	71.0	76.6	57.2	70.2	59.0
RIPT-VLA (Tan et al., 2025)	55.2	31.2	77.6	88.4	91.6	73.5	74.2	68.4
DreamVLA (Zhang et al., 2026b)	26.2	17.6	67.0	77.5	71.5	53.6	43.5	48.9
X-VLA (Zheng et al., 2026)	23.4	89.7	75.7	88.2	96.0	62.7	71.8	71.4
AVA-VLA (Xiao et al., 2026)	69.4	34.9	81.5	**97.5**	94.1	79.1	78.3	74.7
VLA-JEPA (Sun et al., 2026)	63.3	67.1	85.4	95.6	93.6	66.3	85.1	79.5
Cosmos-Policy (Kim et al., 2026b)	75.8	63.3	81.7	96.5	88.9	92.7	82.2	82.2
InternVLA-A1.5 (Ma et al., 2026)	**83.1**	55.1	86.9	96.4	**98.2**	**95.6**	85.2	84.8
$\pi^{\dagger}_{0.5}$ (Intelligence et al., 2025)	70.9	75.2	85.3	96.5	94.3	86.5	85.7	83.9
+ FlowShift	73.4	76.1	85.4	96.4	95.7	87.0	**87.3**	85.0 (+1.1)
RLDX† (Kim et al., 2026a)	72.3	91.0	90.4	94.8	93.9	79.1	81.1	85.3
+ FlowShift	73.8	**91.4**	**91.5**	96.1	94.7	79.0	81.3	**86.0** (+0.7)

Table 2: Performance comparison on RoboTwin 2.0 (Chen et al., 2025). We report success rates (%) over 100 rollouts under the Easy and Hard settings. Representative tasks are shown, while Average is computed over all 50 tasks. † denotes results reproduced by us. Full per-task results are provided in the Appendix. Best results are highlighted in **bold**, with FlowShift gains over $\pi_{0.5}$ shown in green.

Simulation Task	RDT (Liu et al., 2024)		π_0 (Black et al., 2024)		ACT (Zhao et al., 2023)		DP (Chi et al., 2025)		DP3 (Ze et al., 2024)		$\pi^{\dagger}_{0.5}$ (Intelligence et al., 2025)		$\pi_{0.5}$ + FlowShift	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Beat Block Hammer	77	37	43	21	56	3	42	0	72	8	92	42	**97** (+5)	**55** (+13)
Click Bell	80	9	44	3	58	3	54	0	**90**	0	25	52	49 (+24)	**61** (+9)
Place Burger Fries	50	27	80	4	49	0	72	0	72	18	69	66	**86** (+17)	**74** (+8)
Pick Dual Bottles	42	13	57	12	31	0	24	0	60	1	80	30	**85** (+5)	**43** (+13)
...	...		...		...		...		...		...		...	
Place A2B Right	1	1	27	6	0	0	13	0	49	0	65	19	**67** (+2)	**39** (+20)
Place Mouse Pad	1	0	7	1	0	0	0	0	4	1	17	17	**25** (+8)	**31** (+14)
Place Object Scale	1	0	10	0	0	0	1	0	15	0	63	28	**67** (+4)	**44** (+16)
Press Stapler	41	24	62	29	31	6	6	0	69	3	78	47	**83** (+5)	**61** (+14)
Average	34.5	13.7	46.4	16.3	29.7	1.7	28.0	0.6	55.2	5.0	63.9	39.0	**68.9** (+5.0)	**46.2** (+7.2)

for 300 epochs with batch size 64. The routing threshold is calibrated on held-out successful rollouts ($\alpha = 0.1$); see Appendix C.

Results on LIBERO-Plus. We follow the zero-transfer setting, where policies trained on the original LIBERO benchmark are directly evaluated on LIBERO-Plus without perturbation-specific training. As shown in Table 1, FlowShift improves the overall success rate from 83.9% to 85.0% for $\pi_{0.5}$ and from 85.3% to 86.0% for RLDX. The largest gains occur under camera, layout, and background shifts for $\pi_{0.5}$, and camera, lighting, and language perturbations for RLDX. RLDX with FlowShift achieves the highest overall success rate of 86.0%.

Results on RoboTwin 2.0. We follow the standard RoboTwin 2.0 protocol, training each task with 50 clean demonstrations on Aloha-AgileX and evaluating 100 rollouts under both the *Easy* and *Hard* settings. As shown in Table 2, FlowShift improves the average success rate of $\pi_{0.5}$ from 63.9% to 68.9% on Easy and from 39.0% to 46.2% on Hard. The larger gain under Hard supports our selective TTS design, where stochastic refinement is more useful when ODE predictions degrade under distribution shifts. Notable Hard gains include Place A2B Right (+20), Place Object Scale (+16), Press Stapler (+14), and Place Mouse Pad (+14). Per-task results are provided in Appendix D.1.

4.2 Evaluation on Real-World

We evaluate FlowShift on the GALAXEA R1 Lite, a 14-DoF bimanual robot, across *Generalization Tasks* and *Long-Horizon Tasks*. The former evaluates robustness to variations in object positions, appearances, and interchangeable instances, while the latter requires completing dependent subgoals with coordinated bimanual execution. We collect 80 expert demonstrations per task and conduct 20 rollouts with randomized object positions. See Appendix D.2 for details.

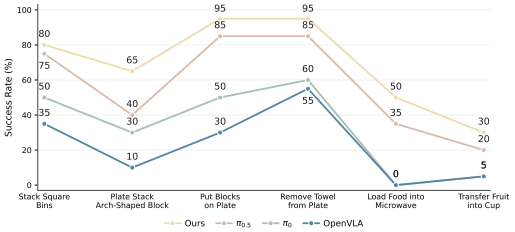

Figure 4: Real-world success rates on six tasks. We compare FlowShift, $\pi_{0.5}$, π_0, and OpenVLA on generalization and long-horizon tasks.

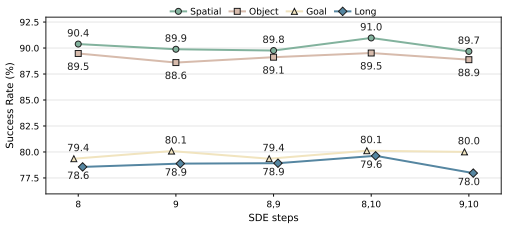

Figure 5: Ablation study of SDE transition schedules on four LIBERO suites. The $\{8, 10\}$ schedule performs best or tied-best overall.

Table 3: Ablation study of the verifier and router. We report success rates (%) on LIBERO-Plus and real-world tasks.

Verifier	Router	LIBERO-Plus					Real-World	
		Spa.	Obj.	Goal	Long	Avg.	Gen.	Long-H.
		89.5	88.7	79.9	78.0	83.9	71.3	27.5
	✓	90.2	89.3	79.4	78.4	84.2	80.0	32.5
✓		89.8	88.6	79.3	78.4	83.9	73.8	30.0
✓	✓	**91.0**	**89.5**	**80.1**	**79.6**	**85.0**	**83.8**	**40.0**

Table 4: Ablation study of verifier adaptation and gradient steering. We report success rates (%) on LIBERO-Plus and real-world tasks.

TTA	Grad.	LIBERO-Plus					Real-World	
		Spa.	Obj.	Goal	Long	Avg.	Gen.	Long-H.
		90.5	88.6	79.3	78.3	84.1	75.0	30.0
✓		89.8	88.9	78.7	78.0	83.7	73.8	27.5
	✓	90.0	88.5	79.1	78.3	83.9	70.0	27.5
✓	✓	**91.0**	**89.5**	**80.1**	**79.6**	**85.0**	**83.8**	**40.0**

Results on Real-World. As shown in Figure 4, FlowShift consistently improves $\pi_{0.5}$ across all six tasks, increasing the average success rate from 71.3% to 83.8% on Generalization Tasks and from 27.5% to 40.0% on Long-Horizon Tasks. We attribute the generalization gains to the test-time adaptive verifier and verifier-guided stochastic sampling: the former adapts task representations to deployment changes, while the latter redirects uncertain action flows toward higher-progress regions. For long-horizon tasks, where errors accumulate across dependent subgoals, the gains highlight the benefit of selective risk-aware refinement for maintaining stable multi-stage execution.

4.3 Ablation Study

Ablation on Verifier and Router. As shown in Table 3, removing the task-progress verifier causes drops of 3.8 and 7.5 points on Real-World Generalization and Long-Horizon tasks, respectively. Without task-progress guidance, the model cannot reliably distinguish actions that truly advance the task from locally plausible but ineffective ones, weakening refinement under deployment shifts. Similarly, removing the risk-aware router leads to 10.0-point drops on both task categories. Without selective routing, TTS may perturb reliable ODE predictions and cause unnecessary shifts.

Table 4 further examines TTA and verifier-gradient steering. TTA adapts the verifier to deployment changes, while gradient steering uses this adapted reward signal to redirect the action flow. Using either alone is insufficient to fully exploit the deployment-aware verifier signal.

Ablation on Router Design. As shown in Table 5, adding flow-timestep aggregation improves RoboTwin 2.0 *Hard* from 44.3% to 45.4%. By aggregating intermediate flow evidence, the router better captures the reliability of the current action generation. Similarly, adding Mamba raises the success rate to 45.1% by modeling cross-chunk execution history. Combining both components further improves the result to 46.2%.

Hyperparameter Analysis. We further study the SDE transition schedule. As shown in Fig. 5, the separated $\{8, 10\}$ schedule achieves the best or tied-best performance across all four LIBERO suites. Adjacent transitions introduce overly concentrated stochastic perturbations, whereas spacing them apart allows intermediate ODE refinement to stabilize the trajectory. This confirms that effective stochastic refinement requires balancing exploration with trajectory preservation.

4.4 Qualitative Analysis

Figure 6 shows a representative LIBERO-Plus example under test-time distribution shift. The pre-trained $\pi_{0.5}$ follows its deterministic ODE trajectory but deviates during the critical placement stage, causing the manipulation to fail. In contrast, FlowShift identifies the unreliable action chunks and se-

Table 5: Ablation study of the router design. We follow the routing setting of SAFE (Gu et al., 2026) as the reference and evaluate timestep aggregation and Mamba-based temporal modeling on LIBERO-Plus, RoboTwin 2.0 Hard, and real-world tasks. Best results are in **bold**.

Timestep Aggregation	Mamba	LIBERO-Plus					RoboTwin Hard	Real-World	
		Spatial	Object	Goal	Long	Average		Generalization	Long-Horizon
		90.3	89.0	79.7	78.5	84.3	44.3	71.3	30.0
	✓	90.4	88.9	79.4	77.8	84.0	45.1	72.5	30.0
✓		90.5	88.4	79.7	78.6	84.2	45.4	76.3	35.0
✓	✓	**91.0**	**89.5**	**80.1**	**79.6**	**85.0**	**46.2**	**83.8**	**40.0**

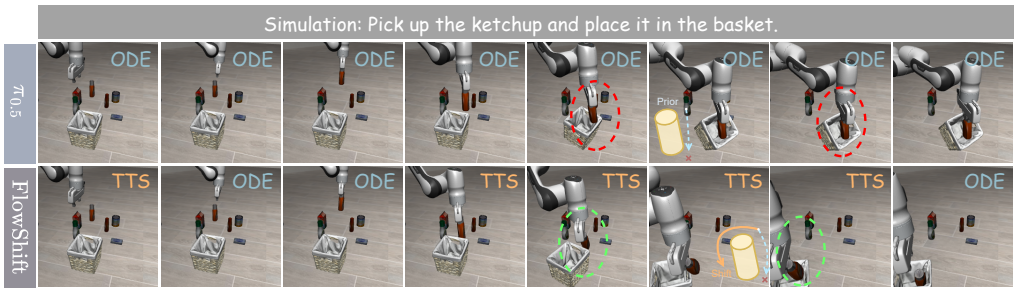

Figure 6: Qualitative comparison on LIBERO-Plus. **Top:** $\pi_{0.5}$ follows the original ODE trajectory and deviates at a critical interaction stage, resulting in task failure. **Bottom:** FlowShift activates TTS at high-risk chunks, correcting the trajectory before resuming ODE.

lectively activates TTS, shifting the action away from the unfavorable prior toward a higher-progress direction. Once the risky interaction is corrected, FlowShift returns to ODE inference and completes the task. More examples are provided in Appendix E.

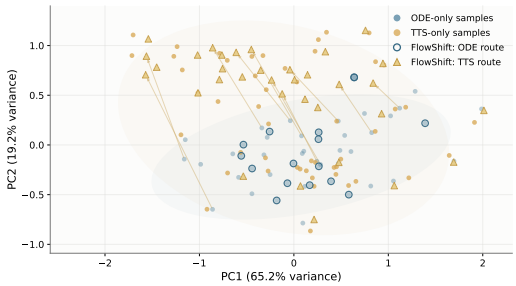

Figure 7: Action distribution shift under FlowShift. TTS shifts samples away from the original ODE region.

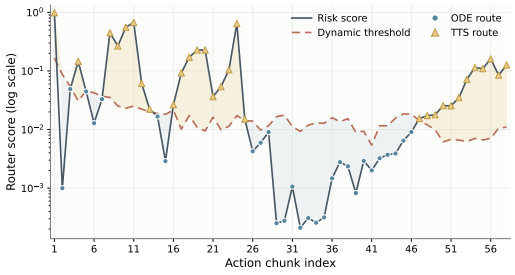

Figure 8: Risk-aware routing during execution. The router switches between ODE and TTS as risk crosses the dynamic threshold.

4.5 Visualization of Risk-Aware Test-Time Scaling

To analyze FlowShift's risk-aware test-time scaling, we visualize action distributions and routing behavior in Figs. 7 and 8. Figure 7 shows that TTS shifts action latents away from the original ODE region, while FlowShift selectively preserves ODE samples or follows the shifted TTS route. Figure 8 shows that the router switches between ODE and TTS as risk crosses the dynamic threshold. Together, these results show that FlowShift regulates distribution shift based on execution risk.

5 Conclusion

In this work, we introduce FlowShift, a plug-and-play, risk-aware test-time scaling framework for improving VLA robustness under deployment shifts. By combining a test-time adaptive task-progress verifier, verifier-guided stochastic flow sampling, and a risk-aware router, FlowShift selectively reshapes intermediate flow dynamics toward reward-aligned action distributions while preserving the pretrained VLA backbone. Extensive experiments demonstrate consistent improvements across multiple simulation benchmarks and real-world tasks, with particularly substantial gains under challenging distribution shifts. These results suggest that selectively adapting inference-time flow dynamics provides a scalable path toward robust VLA deployment in changing environments.

AI USE STATEMENT

In this work, we used generative AI tools solely to aid and polish the writing of the manuscript, including improving language clarity, grammar, and readability. Generative AI was not used to generate research ideas, experimental results, or scientific claims. All AI-assisted text was carefully reviewed and revised by the authors to ensure technical correctness and consistency with our intended meaning. We take responsibility for the final content of this work, including any text produced with the aid of generative AI.

REFERENCES

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control, 2024. *URL https://arxiv. org/abs/2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

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

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: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40041–40076, 2026.

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

Suhyeok Jang, Dongyoung Kim, Changyeon Kim, Youngsuk Kim, and Jinwoo Shin. Verifier-free test-time sampling for vision-language-action models. In *International Conference on Learning Representations*, volume 2026, pp. 139997–140017, 2026.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, et al. Robottt: Context scaling for robot policies. *arXiv preprint arXiv:2607.15275*, 2026.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, et al. Rldx-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026a.

Jaihoon Kim, Taehoon Yoon, Jisung Hwang, and Minhyuk Sung. Inference-time scaling for flow models via stochastic generation and rollover budget forcing. *arXiv preprint arXiv:2503.19385*, 2025a.

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. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025b.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026b.

Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3200–3217. PMLR, 2025.

Weiqi Li, Quande Zhang, Ruifeng Zhai, Liang Lin, and Guangrun Wang. Vla models are more generalizable than you think: Revisiting physical and spatial modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35025–35035, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. Vls: Steering pretrained robot policies via vision-language models. *arXiv preprint arXiv:2602.03973*, 2026.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Haoxiang Ma, Junhao Cai, Xiaoxu Xu, Hao Li, Yuyin Yang, Yang Tian, Jiafei Cao, Hongrui Zhu, Zherui Qiu, Yuqiang Yang, et al. Internvla-a1. 5: Unifying understanding, latent foresight, and action for compositional generalization. *arXiv preprint arXiv:2607.04988*, 2026.

Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *Conference on Robot Learning (CoRL)*, 2024.

Sangwu Park, Wonjoong Kim, Yeonjun In, Sein Kim, Hongseok Kang, and Chanyoung Park. Test-time training for visual foresight vision-language-action models. *arXiv preprint arXiv:2605.08215*, 2026.

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. *arXiv preprint arXiv:2501.09747*, 2025.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. Vla-jepa: Enhancing vision-language-action model with latent world model. *arXiv preprint arXiv:2602.10098*, 2026.

Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive post-training for vision-language-action models. *arXiv preprint arXiv:2505.17016*, 2025.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, Siteng Huang, Yifan Tang, Wenhui Wang, Ru Zhang, Jianyi Liu, and Donglin Wang. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model. *arXiv preprint arXiv:2509.09372*, 2025.

Lei Xiao, Jifeng Li, Juntao Gao, Feiyang Ye, Yan Jin, Jingjing Qian, Jing Zhang, Yong Wu, and Xiaoyuan Yu. Ava-vla: Improving vision-language-action models with active visual attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13453–13463, 2026.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

Siyuan Yang, Yang Zhang, Haoran He, Ling Pan, Xiu Li, Chenjia Bai, and Xuelong Li. Steering vision-language-action models as anti-exploration: A test-time scaling approach. *arXiv preprint arXiv:2512.02834*, 2025.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Wenbo Zhang, Jianxiong Li, Shuai Yang, Sijin Chen, Jiajun Liu, Lingqiao Liu, and Xiao Ma. Ttt-vla: Test-time latent prompt optimization for vision-language-action models. *arXiv preprint arXiv:2606.03127*, 2026a.

Wenyao Zhang, Hongsi Liu, Zekun Qi, Yunnan Wang, Xinqiang Yu, Jiazhao Zhang, Runpei Dong, Jiawei He, He Wang, Zhizheng Zhang, et al. Dreamvla: a vision-language-action model dreamed with comprehensive world knowledge. *Advances in Neural Information Processing Systems*, 38: 24195–24228, 2026b.

Sizhe Zhao, Shengping Zhang, Shuo Yang, Weiyu Zhao, Shuigen Wang, and Xiangyang Ji. Tap-Sampling: Inference-time sampling with a task-progress-understanding verifier for robotic manipulation. In *Forty-third International Conference on Machine Learning*, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.

A Derivations for Verifier-Guided Flow Sampling

This section derives the reward-aligned distribution, stochastic flow transition, clean-endpoint estimation, and verifier-guided proposal introduced in Sec. 3.3. We follow the notation in the main text, where $p_0(\mathbf{A}) = p(\mathbf{A} \mid \mathbf{c}_t; \boldsymbol{\theta})$ denotes the pretrained action distribution and $r(\mathbf{A}) = r_{\boldsymbol{\phi}}(O_t, \ell, \mathbf{A}; \mathbf{P})$ denotes the task-progress reward.

A.1 KL-Regularized Reward Alignment

We first derive the reward-aligned distribution and equation 3. Consider

$$p_{\mathrm{tar}} = \arg\max_{q} \left[\mathbb{E}_{\mathbf{A}\sim q}[r(\mathbf{A})] - \lambda D_{\mathrm{KL}}(q\|p_0)\right], \tag{8}$$

subject to $\int q(\mathbf{A})\, d\mathbf{A} = 1$, where $\lambda > 0$ controls the trade-off between reward maximization and deviation from the pretrained distribution. The corresponding Lagrangian is

$$\begin{aligned}\mathcal{L}(q, \beta) = &\int q(\mathbf{A}) r(\mathbf{A})\, d\mathbf{A} - \lambda \int q(\mathbf{A}) \log \frac{q(\mathbf{A})}{p_0(\mathbf{A})}\, d\mathbf{A} \\ &+ \beta \left(\int q(\mathbf{A})\, d\mathbf{A} - 1 \right).\end{aligned} \tag{9}$$

Its functional derivative is

$$\frac{\delta \mathcal{L}}{\delta q(\mathbf{A})} = r(\mathbf{A}) - \lambda \left(\log \frac{q(\mathbf{A})}{p_0(\mathbf{A})} + 1 \right) + \beta. \tag{10}$$

Setting it to zero gives

$$q^\star(\mathbf{A}) = C\, p_0(\mathbf{A}) \exp\left(\frac{r(\mathbf{A})}{\lambda} \right), \tag{11}$$

where C is independent of $\mathbf{A}$. Normalization then yields

$$\begin{aligned} Z &= \int p_0(\mathbf{A}) \exp\left(\frac{r(\mathbf{A})}{\lambda} \right) d\mathbf{A}, \\ p_{\mathrm{tar}}(\mathbf{A}) &= \frac{1}{Z} p_0(\mathbf{A}) \exp\left(\frac{r(\mathbf{A})}{\lambda} \right). \end{aligned} \tag{12}$$

Thus, reward tilting increases the probability of high-reward actions while retaining the pretrained action prior through p_0.

A.2 Velocity-to-Score Conversion and Same-Marginal SDE

For the linear flow path,

$$\mathbf{A}_t^\tau = \tau \mathbf{A}_t + (1-\tau)\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{13}$$

$$\mathbf{u}\left(\mathbf{A}_t^\tau \mid \mathbf{A}_t\right) = \mathbf{A}_t - \epsilon. \tag{14}$$

Conditioned on $\mathbf{A}_t$,

$$p_\tau\left(\mathbf{A}_t^\tau \mid \mathbf{A}_t\right) = \mathcal{N}\left(\tau \mathbf{A}_t, (1-\tau)^2 \mathbf{I}\right).$$

Its score can therefore be written as

$$\nabla_{\mathbf{A}_t^\tau} \log p_\tau\left(\mathbf{A}_t^\tau \mid \mathbf{A}_t\right) = -\frac{\epsilon}{1-\tau}, \tag{15}$$

$$= \frac{\tau\, \mathbf{u}\left(\mathbf{A}_t^\tau \mid \mathbf{A}_t\right) - \mathbf{A}_t^\tau}{1-\tau}, \tag{16}$$

where the second equality follows from $\tau \mathbf{u}(\mathbf{A}_t^\tau \mid \mathbf{A}_t) - \mathbf{A}_t^\tau = -\epsilon$. Replacing the target vector field with the learned velocity gives

$$\mathbf{s}_{\boldsymbol{\theta}} = \frac{\tau \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t) - \mathbf{A}_t^\tau}{1-\tau}. \tag{17}$$

We next construct an SDE with the same marginal density evolution as the probability-flow ODE

$$\frac{d\mathbf{A}_t^\tau}{d\tau} = \mathbf{v}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \mathbf{c}_t).$$

Its marginal density satisfies

$$\frac{\partial p_\tau}{\partial \tau} = -\nabla_{\mathbf{A}} \cdot \left(p_\tau \mathbf{v}_{\boldsymbol{\theta}}\right). \tag{18}$$

Consider the stochastic process

$$d\mathbf{A}_t^\tau = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t) d\tau + g(\tau) d\mathbf{W}_\tau, \tag{19}$$

whose Fokker–Planck equation is

$$\frac{\partial p_\tau}{\partial \tau} = -\nabla_{\mathbf{A}} \cdot (p_\tau \mathbf{f}_{\boldsymbol{\theta}}) + \frac{1}{2} g(\tau)^2 \Delta_{\mathbf{A}} p_\tau. \tag{20}$$

Using $\mathbf{s}_{\boldsymbol{\theta}} \approx \nabla_{\mathbf{A}} \log p_\tau(\mathbf{A})$, we choose

$$\mathbf{f}_{\boldsymbol{\theta}} = \mathbf{v}_{\boldsymbol{\theta}} + \frac{1}{2} g(\tau)^2 \mathbf{s}_{\boldsymbol{\theta}}. \tag{21}$$

In the exact-score limit,

$$\begin{aligned} &- \nabla_{\mathbf{A}} \cdot (p_\tau \mathbf{f}_{\boldsymbol{\theta}}) + \frac{1}{2} g(\tau)^2 \Delta_{\mathbf{A}} p_\tau \\ &= -\nabla_{\mathbf{A}} \cdot (p_\tau \mathbf{v}_{\boldsymbol{\theta}}) - \frac{1}{2} g(\tau)^2 \nabla_{\mathbf{A}} \cdot (p_\tau \nabla_{\mathbf{A}} \log p_\tau) + \frac{1}{2} g(\tau)^2 \Delta_{\mathbf{A}} p_\tau \\ &= -\nabla_{\mathbf{A}} \cdot \left(p_\tau \mathbf{v}_{\boldsymbol{\theta}}\right), \end{aligned} \tag{22}$$

where $p_\tau \nabla_{\mathbf{A}} \log p_\tau = \nabla_{\mathbf{A}} p_\tau$. Hence, the score-dependent drift exactly compensates for the diffusion term, preserving the ODE marginal evolution. Combining Eqs. equation 17 and equation 21 gives Eq. equation 4.

A.3 Discretization of the Stochastic Flow

Applying Euler–Maruyama discretization to the same-marginal SDE with step size δ, and using $\mathbf{W}_{\tau+\delta} - \mathbf{W}_\tau = \sqrt{\delta}\,\xi$ with $\xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, gives

$$\mathbf{A}_t^{\tau+\delta} = \mathbf{A}_t^\tau + \delta\, \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t) + g(\tau)\sqrt{\delta}\,\xi. \tag{23}$$

Independent noise samples therefore produce distinct local proposals from the same intermediate flow state.

A.4 Clean-Endpoint Estimation

From $\mathbf{A}_t^\tau = \tau \mathbf{A}_t + (1-\tau)\epsilon$ and $\mathbf{u} = \mathbf{A}_t - \epsilon$,

$$\mathbf{A}_t^\tau + (1-\tau)\mathbf{u} = \tau \mathbf{A}_t + (1-\tau)\epsilon + (1-\tau)(\mathbf{A}_t - \epsilon) = \mathbf{A}_t. \tag{24}$$

Replacing $\mathbf{u}$ with the learned velocity gives the clean-endpoint estimate

$$\widehat{\mathbf{A}}_t^\tau = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}_{\boldsymbol{\theta}}\left(\mathbf{A}_t^\tau, \mathbf{c}_t\right), \tag{25}$$

which allows the verifier, defined over action chunks, to evaluate intermediate flow states through their predicted clean endpoints.

A.5 Verifier-Guided Stochastic Proposal

From Eq. equation 3, differentiating the log-density gives

$$\nabla_{\mathbf{A}} \log p_{\mathrm{tar}}(\mathbf{A}) = \nabla_{\mathbf{A}} \log p_0(\mathbf{A}) + \frac{1}{\lambda} \nabla_{\mathbf{A}} r(\mathbf{A}). \tag{26}$$

Thus, reward tilting adds a reward-gradient correction to the score.

At an intermediate flow state, the verifier is applied to the predicted clean endpoint. We therefore define

$$\widetilde{p}_\tau(\mathbf{A}_t^\tau) \propto p_\tau(\mathbf{A}_t^\tau) \exp\left(\frac{r(\widehat{\mathbf{A}}_t^\tau)}{\lambda}\right), \tag{27}$$

whose score is

$$\nabla_{\mathbf{A}_t^\tau} \log \widetilde{p}_\tau(\mathbf{A}_t^\tau) = \mathbf{s}_{\boldsymbol{\theta}} + \frac{1}{\lambda} \nabla_{\mathbf{A}_t^\tau} r\left(\widehat{\mathbf{A}}_t^\tau\right), \tag{28}$$

where the reward gradient is propagated through $\widehat{\mathbf{A}}_t^\tau$.

Substituting this score into the same-marginal drift yields

$$\mathbf{f}_{\mathrm{guided}} = \mathbf{f}_{\boldsymbol{\theta}} + \frac{1}{2\lambda} g(\tau)^2 \nabla_{\mathbf{A}_t^\tau} r\left(\widehat{\mathbf{A}}_t^\tau\right). \tag{29}$$

In practice, we generalize the theoretical coefficient $1/2$ to a time-dependent guidance coefficient $\omega(\tau)$:

$$\mathbf{f}_{\mathrm{guided}} = \mathbf{f}_{\boldsymbol{\theta}} + \frac{\omega(\tau)}{\lambda} g(\tau)^2 \nabla_{\mathbf{A}_t^\tau} r\left(\widehat{\mathbf{A}}_t^\tau\right), \tag{30}$$

where $\omega(\tau) = 1/2$ recovers the drift directly induced by reward tilting.

Finally, Euler–Maruyama discretization with independent $\xi_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ gives

$$\begin{aligned} \mathbf{A}_{t,i}^{\tau+\delta} = \mathbf{A}_t^\tau &+ \delta\, \mathbf{f}_{\boldsymbol{\theta}}\left(\mathbf{A}_t^\tau, \tau, \mathbf{c}_t\right) + g(\tau)\sqrt{\delta}\,\xi_i \\ &+ \frac{\omega(\tau)}{\lambda} g(\tau)^2 \delta\, \nabla_{\mathbf{A}_t^\tau} r\left(\widehat{\mathbf{A}}_t^\tau\right). \end{aligned} \tag{31}$$

The stochastic term generates diverse local particles, while the reward-gradient term biases their proposal distribution toward higher predicted task progress. The resulting particles are then evaluated using the scalar verifier reward as described in Sec. 3.3.

B THEORETICAL DERIVATIONS FOR RISK-AWARE TTS ROUTING

This section provides the theoretical motivation for the risk-aware TTS routing mechanism introduced in Sec. 3.4. Building on the reward-aligned distribution steering derived in Appendix A, we analyze the local benefit of TTS steering, characterize when excessive steering can become detrimental, and show how chunk-wise routing controls trajectory-level deviation from the pretrained ODE policy.

B.1 TTS AS DIRECTED DISTRIBUTION STEERING

As established in Appendix A, verifier-guided TTS realizes KL-regularized distribution steering relative to the pretrained ODE policy. The stochastic flow expands the local candidate set, while the verifier gradient and scalar reward steer and select candidates toward higher predicted task progress. The KL regularization simultaneously discourages excessive deviation from the pretrained action prior.

Therefore, the relevant question is not whether TTS can alter the action distribution, but how much steering is desirable for a given action chunk. We next show that the locally beneficial steering magnitude is generally finite and context dependent.

B.2 FINITE OPTIMALITY OF LOCAL ACTION STEERING

For action chunk c, let $\mathbf{A}_c^{\mathrm{ODE}}$ and $\mathbf{A}_c^{\mathrm{TTS}}$ denote the ODE and TTS predictions, respectively, and define

$$\mathbf{d}_c = \mathbf{A}_c^{\mathrm{TTS}} - \mathbf{A}_c^{\mathrm{ODE}}, \qquad d_c = \|\mathbf{d}_c\|. \tag{32}$$

Vectorization of the action chunk is implicit below.

Let $V(\mathbf{A})$ denote the task value under the current execution context and assume that V is locally L-smooth:

$$\|\nabla_{\mathbf{A}} V(\mathbf{A}_1) - \nabla_{\mathbf{A}} V(\mathbf{A}_2)\| \le L\,\|\mathbf{A}_1 - \mathbf{A}_2\|. \tag{33}$$

The smoothness inequality gives

$$\Delta V_c \ge \nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)^{\top} \mathbf{d}_c - \frac{L}{2} d_c^2, \tag{34}$$

where

$$\Delta V_c = V\left(\mathbf{A}_c^{\mathrm{TTS}}\right) - V\left(\mathbf{A}_c^{\mathrm{ODE}}\right).$$

Define the directional alignment between TTS steering and the local task-value gradient as

$$\rho_c = \frac{\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)^{\top} \mathbf{d}_c}{\|\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)\|\, d_c}, \qquad \rho_c \in [-1, 1], \tag{35}$$

whenever the denominator is non-zero. Then Eq. equation 34 becomes

$$\Delta V_c \ge \rho_c \left\|\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)\right\| d_c - \frac{L}{2} d_c^2. \tag{36}$$

For $\rho_c > 0$, the right-hand side is a concave quadratic in d_c, whose maximum is attained at

$$d_c^{\star} = \frac{\rho_c \left\|\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)\right\|}{L}, \tag{37}$$

with maximum guaranteed improvement

$$\frac{\rho_c^2 \left\|\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)\right\|^2}{2L}. \tag{38}$$

Moreover, the lower bound is positive whenever

$$0 < d_c < \frac{2\rho_c \left\|\nabla_{\mathbf{A}} V\left(\mathbf{A}_c^{\mathrm{ODE}}\right)\right\|}{L}. \tag{39}$$

Thus, even when the steering direction is beneficial ($\rho_c > 0$), the guaranteed gain is non-monotonic in the steering magnitude and is maximized at a finite, context-dependent value. If $\rho_c \le 0$, the same lower bound does not certify improvement for any non-zero steering magnitude.

B.3 When Additional Steering Can Reduce Task Value

We next consider the case where the ODE prediction already lies in a locally high-value region. Suppose V is locally μ-strongly concave:

$$\nabla^2_{\mathbf{A}} V(\mathbf{A}) \preceq -\mu \mathbf{I}, \qquad \mu > 0. \tag{40}$$

Then

$$\Delta V_c \leq \nabla_{\mathbf{A}} V \left(\mathbf{A}_c^{\mathrm{ODE}}\right)^\top \mathbf{d}_c - \frac{\mu}{2} d_c^2. \tag{41}$$

If the ODE prediction is close to a local stationary point,

$$\left\| \nabla_{\mathbf{A}} V \left(\mathbf{A}_c^{\mathrm{ODE}}\right) \right\| \leq \epsilon_c, \tag{42}$$

the Cauchy–Schwarz inequality yields

$$\Delta V_c \leq \epsilon_c d_c - \frac{\mu}{2} d_c^2. \tag{43}$$

Consequently,

$$d_c > \frac{2\epsilon_c}{\mu} \quad \Longrightarrow \quad \Delta V_c < 0. \tag{44}$$

Hence, when the ODE action is already close to a local high-value solution, the available first-order gain is limited and sufficiently large additional steering can move the action away from that region. This motivates avoiding unnecessary TTS activation on already reliable ODE predictions.

B.4 Why TTS Is Allocated to High-Risk Chunks

The preceding analysis shows that the desirable amount of steering depends on the local execution context. Ideally, routing would use the true value improvement ΔV_c, its gradient, and the alignment ρ_c, but these quantities are unavailable at test time. Therefore, the router uses the predicted risk score r_c as an observable proxy for whether the current ODE chunk is likely to benefit from additional correction.

Importantly, r_c is not assumed to directly estimate $\|\nabla_{\mathbf{A}} V\|$ or the distributional deviation. Instead, it summarizes multi-timestep flow features and trajectory history to identify chunks for which verifier-guided steering is more likely to be useful. The routing rule is

$$g_c = \begin{cases} 1, & r_c > \gamma_c, \\ 0, & r_c \leq \gamma_c, \end{cases} \tag{45}$$

where $g_c = 1$ activates TTS and $g_c = 0$ retains ODE decoding.

Thus, risk-aware routing allocates distribution steering to chunks with larger predicted corrective benefit, while preserving the pretrained ODE prediction when additional steering is less likely to be useful.

B.5 Trajectory-Level Distribution Deviation

We now show how chunk-wise routing controls the accumulated deviation from the pretrained trajectory distribution. Let $\mathcal{H}_c$ denote the execution history before chunk c, and let $p_c^{\mathrm{ODE}}(\mathbf{A}_c \mid \mathcal{H}_c)$ and $p_c^{\mathrm{TTS}}(\mathbf{A}_c \mid \mathcal{H}_c)$ denote the corresponding conditional action distributions. The routed distribution is

$$p_c^{g_c}\left(\mathbf{A}_c \mid \mathcal{H}_c\right) = \begin{cases} p_c^{\mathrm{ODE}}\left(\mathbf{A}_c \mid \mathcal{H}_c\right), & g_c = 0, \\ p_c^{\mathrm{TTS}}\left(\mathbf{A}_c \mid \mathcal{H}_c\right), & g_c = 1. \end{cases} \tag{46}$$

For a trajectory $\zeta = (\mathcal{H}_1, \mathbf{A}_1, \ldots, \mathbf{A}_C)$, the routed trajectory distribution factorizes as

$$P_g(\zeta) = p_{\mathrm{init}} \prod_{c=1}^{C} p_c^{g_c}\left(\mathbf{A}_c \mid \mathcal{H}_c\right) \mathcal{T}\left(\mathcal{H}_{c+1} \mid \mathcal{H}_c, \mathbf{A}_c\right), \tag{47}$$

while P_{ODE} is obtained by replacing every $p_c^{g_c}$ with p_c^{ODE}. Since the initial-state distribution and environment dynamics are shared, they cancel in the likelihood ratio:

$$\log \frac{P_g(\zeta)}{P_{\mathrm{ODE}}(\zeta)} = \sum_{c=1}^{C} \log \frac{p_c^{g_c}\left(\mathbf{A}_c \mid \mathcal{H}_c\right)}{p_c^{\mathrm{ODE}}\left(\mathbf{A}_c \mid \mathcal{H}_c\right)}. \tag{48}$$

Taking expectation under P_g gives

$$D_{\mathrm{KL}}\left(P_g \| P_{\mathrm{ODE}}\right) = \mathbb{E}_{P_g}\left[\sum_{c=1}^{C} D_{\mathrm{KL}}\left(p_c^{g_c}\left(\cdot \mid \mathcal{H}_c\right) \| p_c^{\mathrm{ODE}}\left(\cdot \mid \mathcal{H}_c\right)\right)\right]. \tag{49}$$

Define the TTS-induced conditional deviation

$$D_c\left(\mathcal{H}_c\right) = D_{\mathrm{KL}}\left(p_c^{\mathrm{TTS}}\left(\cdot \mid \mathcal{H}_c\right) \| p_c^{\mathrm{ODE}}\left(\cdot \mid \mathcal{H}_c\right)\right). \tag{50}$$

Because the corresponding KL term is zero whenever $g_c = 0$, Eq. equation 49 becomes

$$D_{\mathrm{KL}}\left(P_g \| P_{\mathrm{ODE}}\right) = \mathbb{E}_{P_g}\left[\sum_{c=1}^{C} g_c D_c\left(\mathcal{H}_c\right)\right]. \tag{51}$$

If $D_c(\mathcal{H}_c) \le D_{\max}$ for every chunk, then

$$D_{\mathrm{KL}}\left(P_g \| P_{\mathrm{ODE}}\right) \le D_{\max} \mathbb{E}_{P_g}\left[N_{\mathrm{TTS}}\right], \qquad N_{\mathrm{TTS}} = \sum_{c=1}^{C} g_c. \tag{52}$$

Therefore, each ODE-routed chunk contributes zero additional conditional KL deviation, whereas every TTS activation contributes its corresponding $D_c(\mathcal{H}_c)$. Selective routing thus directly controls the accumulated trajectory-level deviation from the pretrained ODE policy.

B.6 Risk-Aware Routing as a Benefit–Deviation Optimization

The local and trajectory-level analyses lead to a benefit–deviation view of routing. Ideally, the routing policy would maximize the corrective benefit of TTS while constraining its accumulated distributional deviation:

$$\max_{g} \quad \mathbb{E}_{P_g}\left[\sum_{c=1}^{C} \Delta V_c\right] \qquad \text{s.t.} \qquad D_{\mathrm{KL}}\left(P_g \| P_{\mathrm{ODE}}\right) \le \varepsilon. \tag{53}$$

Equivalently, its Lagrangian relaxation is

$$\max_{g} \quad \mathbb{E}_{P_g}\left[\sum_{c=1}^{C} \Delta V_c\right] - \beta_{\mathrm{KL}} D_{\mathrm{KL}}\left(P_g \| P_{\mathrm{ODE}}\right), \qquad \beta_{\mathrm{KL}} > 0. \tag{54}$$

This formulation favors selective TTS activation when corrective benefit varies across chunks. The threshold γ_c provides a practical control over this trade-off: increasing γ_c typically reduces N_{TTS} and therefore tightens the trajectory-deviation bound in Eq. equation 52, while decreasing it permits TTS on more chunks.

Overall, risk-aware routing determines both *where* distribution steering is introduced and *how much* deviation is accumulated over the trajectory, motivating selective TTS activation on high-risk action chunks while retaining ODE decoding elsewhere.

C Implementation and Training Details

This section provides the implementation and training details of the trainable modules in FlowShift. Unless otherwise specified, we use LIBERO-Plus as the representative configuration. The pretrained $\pi_{0.5}$ VLA backbone remains frozen throughout training. For the task-progress verifier, training samples are constructed from the original LIBERO training demonstrations, while the risk-aware router is trained on rollout trajectories collected under the original LIBERO evaluation setting. We describe the architecture and training procedure of these two modules below.

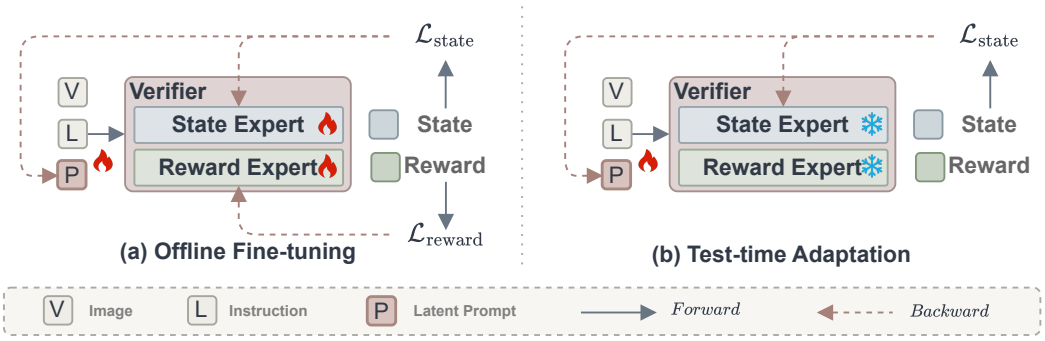

Figure 9: Overview of the test-time adaptive task-progress verifier and its optimization scheme. The Reward Expert estimates task progress, while the State Expert predicts the current proprioceptive state to provide a self-supervised adaptation signal. During offline fine-tuning, the reward and state-prediction objectives follow separate gradient routes. At test time, both experts remain frozen and only the latent prompts are adapted through state prediction.

C.1 Test-Time Adaptive Task-Progress Verifier

C.1.1 Model Architecture

The task-progress verifier is built upon a Prismatic VLM backbone with a hidden dimension of 896. Given the current observation and language instruction, the VLM extracts multimodal task features together with 64 learnable ActionQuery tokens. We further introduce 16 learnable latent-prompt tokens $\mathbf{P}$, which are shared across verifier queries and adapted online at test time. Each candidate action chunk contains 10 actions with an action dimension of 7.

The verifier contains two prediction branches: a **Reward Expert** for task-progress estimation and a **State Expert** for self-supervised latent-prompt adaptation.

The Reward Expert projects each 7-dimensional action into the 896-dimensional hidden space and processes the candidate chunk using 24 Pro MLPResNet blocks with 8-head attention. Each block jointly attends to candidate-action features, ActionQuery representations, and multimodal task features. The Reward Expert predicts a scalar for each action, and the chunk-level task-progress reward is obtained by averaging these predictions over the action horizon.

The State Expert adopts the same 24-block architecture and predicts the current 8-dimensional robot state from a learnable state query. It provides the self-supervised learning signal for latent-prompt adaptation and does not directly contribute to the predicted task-progress reward.

C.1.2 Two-Stage Training

Figure 9 summarizes the offline fine-tuning and test-time adaptation workflow.

For offline fine-tuning, the verifier training samples are constructed from demonstrations in the original LIBERO training data. We fine-tune the verifier for 30,000 optimization steps, while the pretrained $\pi_{0.5}$ policy remains frozen. LoRA is applied to the Prismatic VLM, whereas the Reward Expert, State Expert, ActionQuery tokens, and latent prompts are directly optimized.

Training is divided into two stages. During the first 15,000 steps, the latent prompts are optimized solely through the state-reconstruction objective of the State Expert and are not provided to the Reward Expert. This encourages the prompts to first capture environment-dependent state information.

During the remaining 15,000 steps, the latent prompts are additionally provided to the Reward Expert, allowing the task-progress prediction to incorporate the learned adaptive representation. We apply latent-prompt dropout with a probability of 0.5 during this stage. Reward prediction and state reconstruction both use an ℓ_1 loss, with the state-loss weight set to 1.

We use AdamW with a learning rate of 1×10^{-5} and a weight decay of 0.01. The verifier is trained with a global batch size of 32 in bfloat16 precision on four NVIDIA A800 80 GB GPUs.

C.1.3 TEST-TIME LATENT-PROMPT ADAPTATION

At test time, all verifier parameters are frozen except for the 16 latent-prompt tokens. We maintain an adaptation buffer containing 32 distinct action chunks selected for TTS. Once the buffer is full, the latent prompts are updated using the self-supervised state reconstruction objective before evaluating the current action chunk.

We perform four SGD updates over the buffered samples with a learning rate of 1×10^{-3}. The buffer is then cleared and refilled with subsequent TTS-selected action chunks. This periodic adaptation allows the latent prompts to incorporate information from the deployment environment while keeping the pretrained VLA and the remaining verifier parameters fixed.

C.2 RISK-AWARE ROUTER

C.2.1 MODEL ARCHITECTURE

The risk-aware router operates on intermediate action-decoding features extracted from the frozen $\pi_{0.5}$ policy. We use action features from flow steps $\mathcal{S}_{\text{router}} = \{1, 3, 5, 7\}$. At each selected step, the action features are pooled along the action horizon and then aggregated across flow timesteps to form a chunk-level representation.

To incorporate temporal context across action chunks, the current chunk representation is processed together with the historical chunk sequence using a single-layer Mamba module. The resulting feature is used to predict the chunk-level risk score r_c, which determines whether additional test-time scaling is activated as described in Sec. 3.4.

C.2.2 TRAINING AND CALIBRATION

The router training data are collected through policy rollouts under the original LIBERO evaluation setting. Specifically, we collect rollout trajectories separately on the four original LIBERO suites, with each suite evaluated using a single NVIDIA A800 80 GB GPU.

The router is optimized for 300 epochs with a batch size of 64 and a learning rate of 3×10^{-4}. After training, the routing threshold is calibrated on held-out successful rollout trajectories using conformal prediction with a miscoverage level of $\alpha_{\text{cp}} = 0.1$.

Router training and threshold calibration are conducted on a single NVIDIA A800 80 GB GPU.

D EVALUATION

D.1 EVALUATION ON ROBOTWIN 2.0

RoboTwin 2.0 (Chen et al., 2025) is a large-scale benchmark for bimanual manipulation, built upon the RoboTwin-OD object library with 731 annotated objects from 147 categories. The benchmark contains 50 collaborative manipulation tasks and provides diverse expert trajectories for evaluating robotic policies under varying scene conditions.

For our experiments, we jointly train a $\pi_{0.5}$ policy on all 50 RoboTwin 2.0 tasks using the clean expert demonstrations. Each task contains 50 demonstrations, resulting in a total of 2,500 training demonstrations. At evaluation time, each task is evaluated independently with 100 rollouts under both the *Easy* and *Hard* settings. The *Easy* setting uses relatively clean scenes, whereas the *Hard* setting introduces stronger domain randomization, including object clutter, lighting variations, background textures, and changes in tabletop height.

Results of RDT, π_0, ACT, DP, and DP3 are taken from the official RoboTwin 2.0 benchmark. We reproduce $\pi_{0.5}$ under our joint-training setting and evaluate $\pi_{0.5}$ + FlowShift using the same evaluation protocol. As shown in Table 6, FlowShift improves the average success rate of $\pi_{0.5}$ from 63.9% to 68.9% under the *Easy* setting, and from 39.0% to 46.2% under the *Hard* setting.

Table 6: Complete performance comparison on RoboTwin 2.0. We report per-task success rates (%) over 100 rollouts under the *Easy* and *Hard* settings. † denotes results reproduced by us. The best result for each task and setting is highlighted in **bold**.

Simulation Task	RDT		π_0		ACT		DP		DP3		$\pi_{0.5}^{\dagger}$		$\pi_{0.5}$ + **FlowShift**	
	Easy	*Hard*	*Easy*	*Hard*	*Easy*	*Hard*	*Easy*	*Hard*	*Easy*	*Hard*	*Easy*	*Hard*	*Easy*	*Hard*
Adjust Bottle	81	**75**	90	56	97	23	97	0	99	3	99	31	**100**	37
Beat Block Hammer	77	37	43	21	56	3	42	0	72	8	92	42	**97**	**55**
Blocks Ranking RGB	3	0	19	5	1	0	0	0	3	0	69	48	**76**	**55**
Blocks Ranking Size	0	0	7	1	0	0	1	0	2	0	36	13	**39**	**20**
Click Alarmclock	61	12	63	11	32	4	61	5	**77**	14	61	51	69	**62**
Click Bell	80	9	44	3	58	3	54	0	**90**	0	25	52	49	**61**
Dump Bin Bigbin	64	32	83	24	68	1	49	0	85	53	93	52	**95**	**59**
Grab Roller	74	43	96	80	94	25	98	0	98	2	**99**	93	**99**	**96**
Handover Block	45	14	45	8	42	0	10	0	**70**	0	51	11	50	**19**
Handover Mic	90	31	98	13	85	0	53	0	**100**	3	98	26	**100**	**36**
Hanging Mug	**23**	**16**	11	3	7	0	8	0	17	1	10	8	11	10
Lift Pot	72	9	84	**36**	88	0	39	0	**97**	0	88	4	95	6
Move Can Pot	25	12	58	21	22	4	39	0	**70**	6	51	21	59	**31**
Move Pillbottle Pad	8	0	21	1	0	0	1	0	**41**	0	28	18	36	**23**
Move Playingcard Away	43	11	53	22	36	0	47	0	68	3	68	49	**76**	**54**
Move Stapler Pad	2	0	0	2	0	0	1	0	12	0	23	13	**26**	**20**
Open Laptop	59	32	85	46	56	0	49	0	82	7	89	62	**93**	**65**
Open Microwave	37	20	80	**50**	86	0	5	0	61	22	81	39	**93**	45
Pick Diverse Bottles	2	0	27	6	7	0	6	0	52	1	68	30	**76**	**31**
Pick Dual Bottles	42	13	57	12	31	0	24	0	60	1	80	30	**85**	**43**
Place A2B Left	3	1	31	1	1	0	2	0	46	2	70	**42**	**73**	40
Place A2B Right	1	1	27	6	0	0	13	0	49	0	65	19	**67**	**39**
Place Bread Basket	10	2	17	4	6	0	14	0	26	1	46	41	**47**	**52**
Place Bread Skillet	5	1	23	1	7	0	11	0	19	0	55	**41**	**68**	39
Place Burger Fries	50	27	80	4	49	0	72	0	72	18	69	66	**86**	**74**
Place Can Basket	19	6	41	5	1	0	18	0	**67**	2	42	8	43	**14**
Place Cans Plasticbox	6	5	34	2	16	0	40	0	48	3	**55**	25	**55**	**32**
Place Container Plate	78	17	88	45	72	1	41	0	86	1	90	77	**94**	**87**
Place Dual Shoes	4	4	15	0	9	0	8	0	13	0	40	26	**43**	**36**
Place Empty Cup	56	7	37	11	61	0	37	0	65	1	**94**	66	**94**	**73**
Place Fan	12	2	20	10	1	0	3	0	36	1	62	16	**67**	**22**
Place Mouse Pad	1	0	7	1	0	0	0	0	4	1	17	17	**25**	**31**
Place Object Basket	33	17	16	2	15	0	15	0	65	0	70	23	**71**	**36**
Place Object Scale	1	0	10	0	0	0	1	0	15	0	63	28	**67**	**44**
Place Object Stand	15	5	36	11	1	0	22	0	60	0	80	59	**83**	**64**
Place Phone Stand	15	6	35	7	2	0	13	0	44	2	47	37	**53**	**38**
Place Shoe	35	7	28	6	5	0	23	0	58	2	68	62	**76**	**73**
Press Stapler	41	24	62	29	31	6	6	0	69	3	78	47	**83**	**61**
Put Bottles Dustbin	21	4	54	13	27	1	22	0	60	21	62	52	**66**	**61**
Put Object Cabinet	33	18	68	18	15	0	42	0	**72**	1	27	21	32	**25**
Rotate QRCode	50	5	68	15	1	0	13	0	74	1	78	22	**82**	**26**
Scan Object	4	1	18	1	2	0	9	0	31	1	27	14	**33**	**20**
Shake Bottle	74	45	97	60	74	10	65	8	98	19	99	95	**100**	**98**
Shake Bottle Horizontally	84	51	99	51	63	4	59	18	**100**	25	99	96	**100**	**99**
Stack Blocks Three	2	0	17	0	0	0	0	0	1	0	60	24	**68**	**32**
Stack Blocks Two	21	2	42	1	25	0	7	0	24	0	**87**	48	86	**53**
Stack Bowls Three	51	17	66	24	48	0	63	0	57	5	64	55	**74**	**63**
Stack Bowls Two	76	30	91	41	82	0	61	0	83	6	89	75	**95**	**80**
Stamp Seal	1	0	3	4	2	0	2	0	18	0	**47**	31	46	**35**
Turn Switch	35	15	27	23	5	2	36	1	**46**	8	35	23	45	**33**
Average	34.5	13.7	46.4	16.3	29.7	1.7	28.0	0.6	55.2	5.0	63.9	39.0	**68.9**	**46.2**

D.2 Evaluation on Real-World

Robot Platform. We conduct real-world experiments on the Galaxea R1 Lite dual-arm robotic platform. The policy controls two 6-DoF robotic arms, each equipped with a parallel gripper with discrete open/close commands. All policy inference is deployed on an NVIDIA GeForce RTX 5090 GPU. The policy predicts action chunks from the current observations, which are subsequently transmitted to the robot controller for execution.

Tasks and Demonstration Collection. As summarized in Table 7, we organize the real-world evaluation into two major categories: *Generalization Tasks*, which are designed to evaluate the robustness and generalization ability of the learned policy under distribution shifts, and *Long-Horizon Tasks*, which focus on evaluating the capability of the policy to accomplish temporally extended manipulation tasks involving multiple consecutive stages. Representative task setups are illustrated in Fig. 10 and Fig. 11, respectively. For each task, we collect 80 expert demonstrations. The demonstration set consists of 40 in-distribution (ID) demonstrations under the basic task configuration and 40 out-of-distribution (OOD) demonstrations with deliberately introduced variations. The OOD configurations cover a diverse range of perturbations, including, but not limited to, changes in object layouts and positions, variations in object colors, replacement of objects with different instances from the same semantic category, and the introduction of additional distractor objects. By incorporating both ID and OOD demonstrations during data collection, we aim to expose the policy to diverse task configurations while still retaining a well-defined nominal task distribution.

Evaluation Protocol. At test time, each task is evaluated over 20 independent real-robot trials, with most test scenes configured to be OOD relative to the basic ID settings. The evaluation introduces variations in object placement, appearance, instance identity, and distractors to assess generalization to unseen real-world configurations. Generalization tasks focus on robustness under visual and spatial distribution shifts, while long-horizon tasks further evaluate the ability to complete multiple consecutive manipulation stages. Quantitative results are reported in Table 7, with representative task setups shown in Fig. 10 and Fig. 11.

E CASE STUDY

Table 7: Overview of Generalization and Long-Horizon Tasks on Real-World.

Task name	Generalization Tasks				Long-Horizon Tasks	
	Stack Square Bins	Remove Towel from Plate	Stack Arch-Shaped Block	Put Blocks on Plate	Load Food into Microwave	Transfer Fruit into Cup
Description	Stack one square bin on top of the other square bin.	Remove the towel from the plate.	Stack the arch-shaped block on top of the two cube blocks.	Put all the blocks on the table into the plate.	Put all the food models on the table into the pot, place the pot inside the microwave, and finally close the microwave door.	Use the right arm to place the fruit near the cup, then use the left arm to put the fruit into the cup.
# demonstrations	80	80	80	80	80	80
# rollouts	20	20	20	20	20	20
Performance (%)	80	95	65	95	50	30

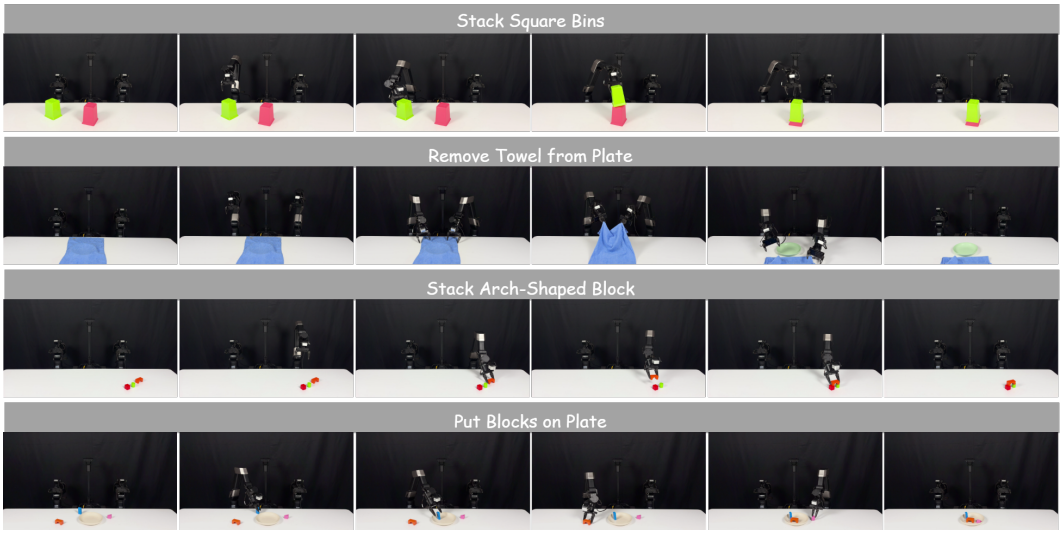

Figure 10: Qualitative visualization of the **generalization tasks**.



Figure 11: Qualitative visualization of the **long-horizon tasks**.

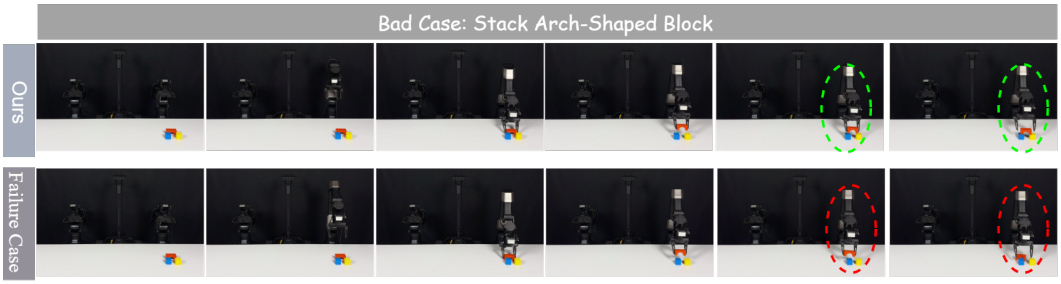

Figure 12: Qualitative visualization of the **generalization tasks**.