FLOW-BASED POLICY ADAPTATION WITHOUT POLICY UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation models and other policies trained on large and diverse datasets can endow robots with semantic and task knowledge that would be difficult to acquire from task-specific demonstrations alone. Despite their broad scene understanding and reasoning capabilities, however, these models often lack the environment- and embodiment-specific knowledge required for precise and reliable execution. We propose GLOVES, a flow-based framework that complements an existing agent with a lightweight, goal-agnostic expert action prior that refines the action chunks proposed by the agent without updating its parameters. The prior captures observation-conditioned expert behavior, while the agent's proposal supplies task- and context-specific intent that the correction aims to preserve. The learned flow also provides a natural in-distribution scoring mechanism through reverse flow evaluation. We use this signal as an intervention gate: actions that appear consistent with the expert distribution are passed through unchanged, while anomalous or out-of-distribution (OOD) actions are corrected. In this way, GLOVES only provides assistance when necessary. By learning local goal-agnostic expert action patterns and using them to adapt agent-proposed actions during execution, GLOVES provides a lightweight shared-control module for robust action adaptation across tasks and environments. Across controlled OOD-recovery studies, simulation benchmarks, and real-robot experiments, GLOVES consistently improves the success of vision-language-action models (VLAs) and Code-as-Policy agents without retraining the underlying policy.

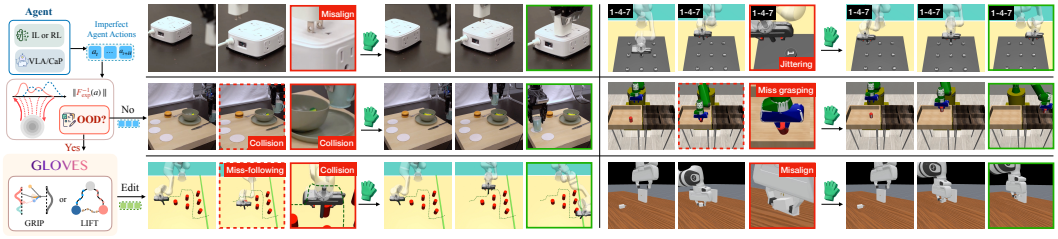


Figure 1: Overview of GLOVES. At inference time, an agent-proposed action chunk is passed to the OOD gate to determine whether the chunk should be executed directly or adapted. When intervention is needed, GLOVES refines the action with GRIP or LIFT and applies the refined chunk for the robot's rollout.

1 INTRODUCTION

Modern robotic systems increasingly rely on large pretrained policies that bring broad prior knowledge to new tasks, particularly vision-language-action (VLA) models trained on large and diverse datasets. With knowledge of the goal or intent (e.g., conveyed via language), these models provide high-level reasoning capabilities that would be difficult to acquire from task-specific demonstrations alone. Yet these capabilities alone are not sufficient to ensure reliable execution—the resulting actions are often noisy, biased, delayed, or invalid due to domain shift (Jones et al., 2025). Typically, these errors are addressed by finetuning the policy using additional task- and environment-specific demonstrations (Kim et al., 2024; 2025b; Team et al., 2024; Myers et al., 2024). However, finetuning can be prohibitively expensive or simply not possible when the agent is only accessible via a

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.

black-box API. Even when the parameters can be updated, incorporating task-specific expertise directly into the policy requires modifying a model whose broad pretrained capabilities we may wish to preserve. We therefore ask: *Can an existing agent's execution be improved through action-level correction, without any policy updates?*

Viewed through the lens of shared autonomy, we formulate this problem as image-conditioned action-chunk adaptation (Zhao et al., 2023; Chi et al., 2023; Janner et al., 2022) and propose GLOVES (Figure 1), a flow-based framework for adapting an existing agent's actions. GLOVES uses goal-agnostic expert demonstrations to learn a lightweight, image-conditioned expert action prior. Critically, unlike the agent, which is conditioned on the task goal (e.g., specified via language), GLOVES does not assume access to the goal during action adaptation. Because the same observation can admit multiple valid behaviors, the agent's proposed chunk provides an essential cue to the goal-dependent behavior it intends to execute (Yoneda et al., 2023; Sun et al., 2025a;b). GLOVES refines this proposal using the learned prior while regularizing toward the original chunk.

GLOVES performs this adaptation through a generative action editing process, using the expert prior to steer the agent's proposed action toward expert behavior while preserving the agent's intent. Balancing these potentially competing objectives is non-trivial. Existing generative editing methods typically sample actions through a noising-denoising process (Meng et al., 2022; Yoneda et al., 2023), where the noise level controls the trade-off between the *fidelity* of the resulting action and its *conformity* to expert behavior. GLOVES provides two strategies for dealing with this tradeoff. Guided Refinement with an Intent-Preserving Energy (GRIP) combines partial noising with energy-guided generation, requiring only an expert flow. Learned Inversion-Free Transport (LIFT) additionally learns an agent flow and applies FlowAlign-style source-to-expert transport (Lipman et al., 2023; Liu et al., 2023; Kim et al., 2025a) without selecting a partial-noising depth. Both strategies regularize the correction toward the agent's original proposal.

GLOVES uses the reverse direction of the trained flow model as an in-distribution scoring mechanism and only adapts actions that are identified as anomalous or out-of-distribution (OOD). Action chunks that pass through this *flow-based OOD gate* are executed unchanged. The same expert flow thus supports both OOD detection and action correction, without training a separate detector. In contrast, existing methods (Yoneda et al., 2023; Reddy et al., 2018; Sun et al., 2025a) either adapt *every* action chunk, at the cost of repeated ODE integrations and potentially overriding valid agent behavior, or they require training a separate uncertainty model in order to decide when to provide assistance (McMahan et al., 2024).

Our contributions are threefold: (1) **Goal-agnostic action adaptation without policy updates.** We introduce GLOVES, a lightweight framework that combines an expert action prior with agent-proposed chunks through GRIP and LIFT. The same expert flow supports selective intervention and action correction without a separately trained detector. (2) **Improved execution for VLA and CaP agents.** We demonstrate action adaptation for both vision-language-action (VLA) and code-as-policy (CaP) agents across three simulation tasks while keeping the underlying agents unchanged. On Slalom, LIFT raises VLA success from 33.06% to 76.67% and CaP success from 63.67% to 97.67%. Real-robot VLA experiments demonstrate complementary capabilities: fine-grained correction for precise charger insertion and sustained trajectory refinement for smooth cup transport. On charger insertion, GRIP increases success from 45% $(18/40)$ to 100% $(40/40)$. (3) **Controlled analysis of detection and correction.**We complement end-to-end evaluation with structured perturbations that provide known injection events for statistical analysis of OOD detection and task-performance recovery across error types. Comparisons with and without gating, together with computational profiling, characterize the benefits and costs of selective intervention.

2 Related Work

Generative Models for Imitation Learning Generative policies are well suited for imitation learning, where demonstrations can exhibit multimodal action distributions when tasks can be performed in different ways. Prior work models this multimodality using implicit discrete sequence models (Shafiullah et al., 2022; Zhao et al., 2023). Diffusion models formulate planning and control as conditional denoising, enabling trajectory-level behavior synthesis (Janner et al., 2022), offline decision making (Ajay et al., 2023), and visuomotor imitation (Reuss et al., 2023; Chi et al., 2023; Ze et al., 2024).

The iterative sampling required by diffusion policies can be costly for closed-loop control, motivating single- or few-step distillation methods such as Consistency Policy (Prasad et al., 2024) and One-Step Diffusion Policy (Wang et al., 2025). Flow-matching policies offer an alternative by learning a velocity field that maps a source distribution to the target distribution, with applications to robot motion learning, manipulation, and generalist robot policies (Braun et al., 2024; Zhang et al., 2025; Jung et al., 2025; Black et al., 2025; Yan et al., 2025). Whereas these methods primarily use generative models to synthesize actions from noise, GLOVES instead adapts agent-proposed actions, selectively moving them toward the expert distribution, while preserving the agent's intent.

Action Adaptation Policy adaptation aims to improve pretrained robot policies (e.g., those trained via imitation learning (IL) or Reinforcement Learning (RL)), including VLA models (Kim et al., 2024; 2025b; Pan et al., 2026; Myers et al., 2024). Some methods directly finetune VLAs (Collaboration et al., 2023; Wang et al., 2024; Team et al., 2024; Black et al., 2025) or adapt them through language-level decomposition (Myers et al., 2024; Shi et al., 2024), while others add failure prediction (Xu et al., 2025), corrective supervision (Yang et al., 2026), or online post-training (Xu et al., 2024) to improve deployment performance. Closest to our setting are residual or correction-based methods that keep the base policy intact and learn an auxiliary module to refine its proposed actions (Yuan et al., 2024; Welte et al., 2026). Related, generative shared-autonomy methods use diffusion (Yoneda et al., 2023) and consistency models (Sun et al., 2025a) to refine actions proposed by a human rather than a pretrained policy while seeking to preserve intent. Selective-intervention methods further study when assistance should be applied (McMahan et al., 2024; Yu et al., 2025), reduce unnecessary assistance, only intervene when the copilot is expected to improve performance. Our approach combines expert-prior-guided editing of action chunks with selective intervention: the agent's proposed action chunk conveys intent, while the same expert flow supports both OOD scoring and editing.

3 PROBLEM FORMULATION

Let I denote the image observation, and g the task goal. Given the goal, an existing agent proposes an action chunk $\text{Agent} : (I, g) \mapsto a_u \in \mathbb{R}^{T \times d_a}$, where T is the chunk length and d_a the action dimension. The adapter receives the proposed action chunk a_u and image observation I. It produces a refined chunk according to

$$\tilde{a} \sim \mathcal{A}_\theta(\cdot \mid a_u, I). \tag{1}$$

We learn an expert action prior from context-action pairs $\mathcal{D}_e = \{(I_i, a_i^e)\}$. Explicit **goal** conditioning is **excluded** from both prior learning and action correction, although the demonstrations may be collected by a goal-informed expert. The agent's proposal therefore serves as a cue to the intended behavior. Our objective is to improve task execution while preserving this intent, while not modifying the underlying agent. We exclude explicit goal conditioning from both prior learning and action correction to decouple execution refinement from the agent's goal representation, which may take the form of language, target states, or images. This allows the expert prior to support agents with different goal interfaces without requiring a shared goal encoder or goal-labeled correction data. The base agent remains responsible for interpreting the task, while its proposed action chunk supplies the adapter with an intent cue.

3.1 PRELIMINARY: FLOW MATCHING

Flow matching (Lipman et al., 2023; Kim et al., 2025a) is a generative modeling framework that learns to draw samples from a target data distribution $\mathcal{P}_1$ by continuously transporting samples drawn from a tractable source distribution $\mathcal{P}_0$ (e.g., a standard Gaussian). Concretely, it transports a sample $x_0 \sim \mathcal{P}_0$ to a sample $x_1 \sim \mathcal{P}_1$ by solving the ODE

$$\frac{dx_t}{dt} = u_t(x_t), \quad t \in [0, 1]. \tag{2}$$

Flow matching learns a neural approximation $v_\theta(x, t)$ for the true velocity field $u_t(x_t)$ parameterized by θ. The ideal flow-matching objective regresses the model prediction to the ground-truth velocity $u_t(x)$ of a probability path $\mathcal{P}_t$ connecting $\mathcal{P}_0$ and $\mathcal{P}_1$:

$$\min_\theta \ \mathrm{E}_{t \sim \mathcal{U}(0,1),\, x_t \sim \mathcal{P}_t} \left[\| v_\theta(x_t, t) - u_t(x_t) \|_2^2 \right]. \tag{3}$$

Since u_t is generally intractable (Lipman et al., 2023), conditional flow matching instead supervises the model with velocities from conditional paths between paired or coupled samples (x_0, x_1). In the rectified-flow setting (Liu et al., 2023), the conditional path is the linear interpolation

$$x_t = (1-t)x_0 + tx_1, \qquad u_t(x_t \mid x_0, x_1) = x_1 - x_0. \tag{4}$$

Training then reduces to solving a supervised regression problem over sampled times and pairs,

$$\min_\theta \ \mathrm{E}_{t,x_0,x_1}\left[\|v_\theta((1-t)x_0 + tx_1, t) - (x_1 - x_0)\|_2^2\right]. \tag{5}$$

Given an external condition c, such as an observation, the target distribution becomes $\mathcal{P}_1(\cdot \mid c)$, and the learned velocity field is written as $v_\theta(x, t \mid c)$. The training objective has the same form as above, with the expectation taken over c and samples from the corresponding conditional path. Integrating $v_\theta(\cdot, t \mid c)$ from $t = 0$ to $t = 1$ defines the conditional flow map $F_\theta(\cdot; c)$.

4 When to adapt: Out-of-Distribution Detection

We use the reverse velocity of the trained expert flow to construct a lightweight out-of-distribution (OOD) score, inspired by flow-based failure detection (Xu et al., 2025).

Given an agent-proposed action chunk a_u and the corresponding image observation I_u, the reverse flow $F_\theta^{-1}(a_u; I_u)$ provides a latent representation associated with the Gaussian base distribution. We approximate this representation from flow time $t = 1$ (target) to $t = 0$ (Gaussian), and define the nonconformity score s as

$$\hat{z}(a_u; I_u) = a_u - v_\theta(a_u, 1 \mid I_u), \qquad s(a_u; I_u) = \|\hat{z}(a_u; I_u)\|_2^2. \tag{6}$$

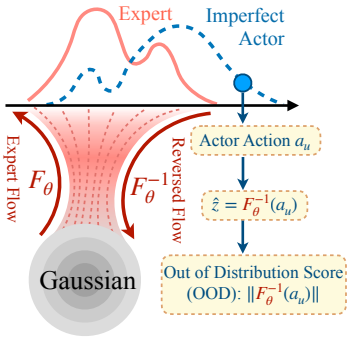


Figure 2: OOD detection.

Larger scores are treated as larger nonconformity to the expert distribution. To test whether action a_u is OOD, we calibrate this score using held-out expert examples $\mathcal{D}_{\text{cal}} = \{(a_i, I_i)\}_{i=1}^n$. Writing $s_i = s(a_i; I_i)$, (notably, I_u does not have to be from $\{I_i\}$) we compute the conformal p-value

$$p(a_u; I_u) = \frac{1 + \sum_{i=1}^n \mathbf{1}\{s_i \geq s(a_u, I_u)\}}{n+1}. \tag{7}$$

A small p-value indicates that the proposed chunk has an unusually large score relative to the expert reference data. When OOD gating is enabled, we apply action correction if $p(a_u; I_u) \leq \alpha$ and otherwise execute the proposed chunk unchanged. The threshold α controls gating sensitivity, with larger values triggering more frequent intervention. We precompute expert calibration scores before deployment, so online gating requires only a single- or few-step reverse-flow evaluation and comparison against the cached scores, supporting real-time intervention decisions. As a check on the score itself, using it alone to guide a local search over action chunks improves task success on all three simulation tasks (Appendix D.2, Table 14).

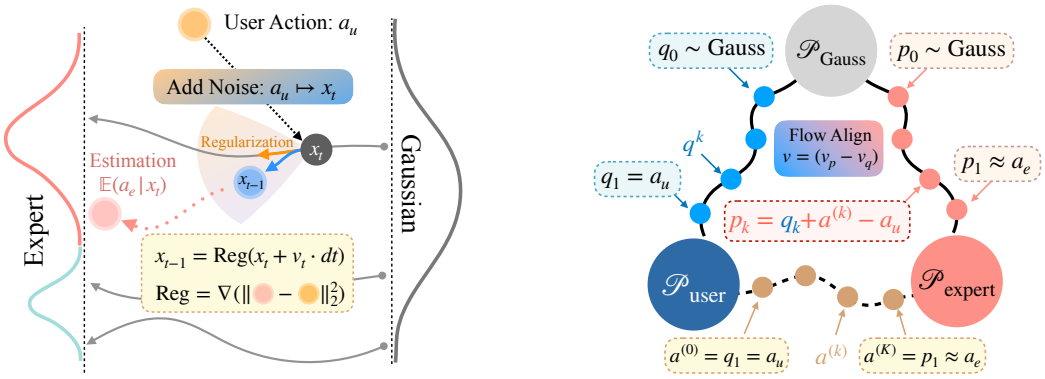
5 How to adapt: Gloves

5.1 Grip: Single-Flow Action Editing

Inspired by inversion-based sample editing (Yoneda et al., 2023; Sun et al., 2025a; Meng et al., 2022), we present **Guided Refinement with an Intent-Preserving Energy** (Grip), which directly guides the learned conditional expert flow to generate expert-like actions that are consistent with the agent's intent. Grip optionally adopts OOD detection to skip editing when the agent's action is already in the expert distribution. Specifically, during inference, we initialize the trajectory from a noisy version of a_u

$$x_{t_{\text{init}}} = t_{\text{init}} a_u + (1 - t_{\text{init}})\epsilon, \qquad \epsilon \sim \mathcal{N}(0, \mathbf{I}), \tag{8}$$

where $t_{\text{init}} \in [0, 1]$ controls the autonomy level. With larger values, Grip adheres more closely to the agent's actions, whereas smaller values resample more aggressively from the expert prior.



(a) GRIP (b) LIFT

Figure 3: Overview of the policy adaptation methods in GLOVES. (a) GRIP follows the expert flow while regularizing toward the original agent command. (b) LIFT uses both agent and expert flows to transport agent actions toward the expert distribution without explicit inversion.

To further preserve the agent's intent, we use energy-guided flow matching (Feng et al., 2025), with an energy function that penalizes deviation from the agent's original proposed action:

$$E(x_1) = \frac{1}{2} \|x_1 - a_u\|_2^2 . \tag{9}$$

During integration, GRIP evaluates this energy function using a one-step Euler estimate $\hat{x}_1$ of the terminal action x_1 under the expert flow:

$$\hat{x}_1(x_t, t) = x_t + (1 - t) v_\theta(x_t, t \mid I), \tag{10}$$

GRIP calculates the refined action chunk $\tilde{a}$ by integrating the guided velocity field $\tilde{v}$ from $t = t_{\text{init}}$ to $t = 1$:

$$\tilde{v}(x_t, t \mid I) = v_\theta(x_t, t \mid I) - \lambda_{\text{align}} \nabla_{x_t} E(\hat{x}_1(x_t, t)), \tag{11}$$

where $\lambda_{\text{align}} \geq 0$ controls guidance strength. For GRIP, across all experiments, we find that $\lambda_{\text{align}} = 1$ yields stable and effective performance.

5.2 LIFT: DUAL-FLOW ACTION EDITING

Learned Inversion-Free Transport (LIFT) extends the expert-only formulation by additionally learning a flow from agent rollouts. Following FlowAlign (Kulikov et al., 2025; Kim et al., 2025a), it edits action chunks using the difference between agent and expert-guided velocity fields, with a source-consistency regularizer based on their estimated clean endpoints. This replaces partial noising with source-to-expert transport and removes the need to choose a partial-noising depth.

Let v_u and $v_e \equiv v_\theta$ denote the image-conditioned velocity fields learned from agent rollouts (v_u) and expert demonstrations (v_e), respectively (Figure 3(b)). We maintain an editing state $a^{(k)}$ on the time grid $0 = t_0 < \cdots < t_K = 1$(similar to local search in sec D.2), initialized at $a^{(0)} = a_u$. At editing step $k = 0, \ldots, K-1$, we sample $\epsilon_k \sim \mathcal{N}(0, I)$ and construct the agent intermediate sample $q_k = (1 - t_k)\epsilon_k + t_k a_u$. We then construct the corresponding expert intermediate sample by shifting q_k according to the current editing state: $p_k = q_k + \left(a^{(k)} - a_u\right)$.

At inference step k (corresponding timestep t_k), we evaluate the source velocity $v_q = v_u(q_k, t_k \mid I)$ and the expert-guided target velocity with classifier-free guidance (CFG)

$$v_p = v_u(p_k, t_k \mid I) + \omega \left[v_e(p_k, t_k \mid I) - v_u(p_k, t_k \mid I) \right], \tag{12}$$

where ω is the CFG scale that controls the expert guidance strength. The difference $(v_p - v_q)$ gives the inversion-free editing direction from the agent action chunk toward the expert action chunk.

To encourage source consistency, LIFT computes one-step clean-action estimates from the source and guided target velocities:

$$\hat{a}_u = q_k + (1 - t_k) v_q, \qquad \hat{a}_e = p_k + (1 - t_k) v_p. \tag{13}$$

We combine the velocity-difference update with an endpoint-consistency correction that pulls the target estimate toward the source estimate:

$$a^{(k+1)} = a^{(k)} + \eta_{\text{edit}}\, \Delta t_k (v_p - v_q) + \lambda_{\text{align}}\, (\hat{a}_u - \hat{a}_e)\,, \tag{14}$$

where $\Delta t_k = t_{k+1} - t_k$, η_{edit} is the edit scale, and λ_{align} controls the endpoint-consistency correction applied at each iteration. After K editing steps, we output $\tilde{a} = a^{(K)}$. The full algorithm is provided in Appendix Algorithm 1. Although LIFT introduces additional hyperparameters, the screening sweep shows less than 2 percentage points of variation in marginal mean success when the hyperparameters are chosen within the following ranges $\omega \in \{0.2, 0.5\}$, $\eta_{\text{edit}} \in \{0.3, 0.5, 0.7\}$, and $\lambda_{\text{align}} \in \{0.05, 0.07, 0.10\}$ (Appendix C.4).

6 EXPERIMENTS

Experimental rationale

1. Does GLOVES improve existing agents? Our primary evaluation demonstrates improved task success for VLA and CaP agents in simulation, and for the VLA on real robots. These gains come from correcting proposed action chunks while keeping the underlying agents unchanged.

2. Why look beyond end-to-end success? Many approaches can improve task success, but success alone does not establish whether OOD action proposals are detected and corrected before execution, or whether the agent simply recovers from errors later. Natural agent rollouts lack per-action ground truth correctness annotations, motivating a controlled perturbation study.

3. How do we evaluate detection and correction? We inject known perturbations into imitation-learning agents' action chunks. Injection events provide labels for evaluating detection through precision, recall, and AUROC; task success with and without correction and OOD gating measures performance recovery across perturbation types and random seeds. These labels identify injected changes, which need not always produce harmful actions.

Our primary evaluation asks whether action correction improves the execution of two distinct agent classes—a FLOWER-based VLA(Reuss et al., 2025) and a code-as-policy (CaP) (Liang et al., 2023) agent—without updating the underlying agents. We evaluate agent adaptation on three simulation tasks and further study VLA adaptation on two real-robot tasks.

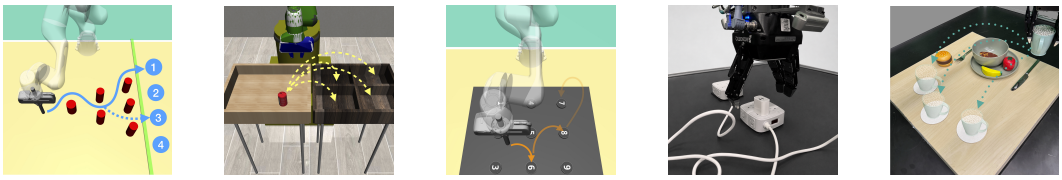
6.1 EXPERIMENTAL SETUP

Tasks and agents. We evaluate three simulation tasks: Slalom, CAN, and Keypad, using both FLOWER (Reuss et al., 2025)-based VLA and CaP agents. We additionally evaluate the FLOWER-based VLA on two real-robot tasks with a UR5: charger insertion and cup transport around an obstacle (Figure 4).

Expert priors and baselines. The expert flows are trained on observation–action pairs **without explicit goal labels**, and action correction also **excludes** explicit goal conditioning. The base agent retains its goal or language instruction, while its proposed action chunk supplies the adapter with an intent cue. We compare against unassisted execution, direct sampling from the goal-agnostic expert flow, *To the Noise and Back* (Yoneda et al., 2023) (DDPM), and *FlashBack CSA* (Sun et al., 2025a), adapted to our image-conditioned action-chunk setting.

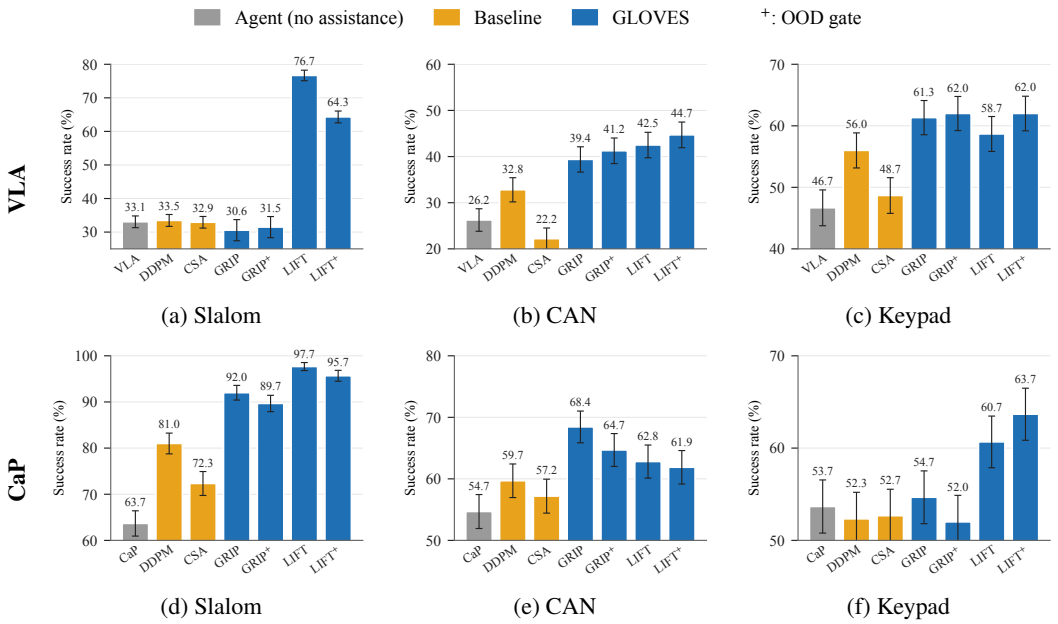
6.2 AGENT ADAPTATION IN SIMULATION AND ON REAL ROBOTS

Simulation with VLA and CaP agents. Figure 5 reports execution before and after action correction in simulation. On Slalom, LIFT raises the FLOWER agent's success from 33.06% to 76.67% and the CaP agent's from 63.67% to 97.67%. For the VLA on Slalom, DDPM, CSA, and GRIP remain within 2.5 percentage points of unassisted execution. Although all three tasks admit multiple goal-dependent behaviors, Slalom's route choice can be only weakly expressed in a short local action chunk, making its intent cues vulnerable to partial noising. In CAN and Keypad, local motion often provides a clearer cue to the immediate manipulation target or button, even without revealing the full task goal. The Slalom results suggest that LIFT better retains these weak intent cues through source-to-expert transport, without partially noising the editing state. On CAN, GRIP gives the



(a) Slalom (b) CAN (c) Keypad (d) Real Charger (e) Serve Cup

Figure 4: Experimental tasks. (a) Slalom navigation with 24 corridor configurations; (b) CAN placement with four target regions; (c) three-button code entry on a 3×3 keypad; (d) real-robot charger insertion; and (e) real-robot cup serving with three coaster targets.



(a) Slalom (b) CAN (c) Keypad

(d) Slalom (e) CAN (f) Keypad

Figure 5: Simulation success rates (%) before and after action correction for the FLOWER-based VLA (top) and the CaP agent (bottom). Error bars show the bootstrap standard error over pooled episodes. $^{+}$: OOD gating. The two agents have different no-assistance baselines, so bars should be compared within each panel.

largest gain for CaP (54.69% to 68.44%), while LIFT with OOD gating gives the largest gain for the VLA (26.25% to 44.70%). On Keypad, GRIP and LIFT with OOD gating both raise the VLA from 46.67% to 62.00%; for CaP, LIFT provides the largest gain, reaching 63.67% with gating compared with 53.67% unassisted. Across the six agent–task pairs, the better GLOVES editor exceeds every evaluated baseline. These gains are obtained by editing the proposed action chunks while leaving the underlying agent unchanged.

VLA adaptation on real robots. Table 1 reports the real-world evaluation, which uses the FLOWER-based VLA. The two tasks examine complementary scales of correction: charger insertion requires high-precision alignment near contact, whereas cup transport requires sustained corrections to maintain a smooth trajectory around an obstacle and reach the instructed target. On charger insertion, GRIP improves success from 18/40 (45%) to 40/40 (100%); CSA and LIFT achieve 21/40 and 27/40, respectively. On cup transport, successful trials increase from 2 for the base VLA to 22 with GRIP and 16 with LIFT, compared with 4 for CSA. Together, these experiments demonstrate precise local correction and sustained trajectory refinement during real-robot execution.

Table 1: Real-robot success of the FLOWER-based VLA (successful/total trials). Bold: best per task.

Task	Unassisted	CSA	GRIP	LIFT
Charger Insertion	18/40	21/40	**40/40**	27/40
Cup Serve	2/30	4/30	**22/30**	16/30

Table 2: One-step OOD detection by wrapper.

Wrapper	Precision	Recall	AUROC
Noised	.997	1.000	1.000
Laggy	.913	.344	.780
Slow	.997	.931	.988
Shift	.994	.756	.983
Macro	.975	.758	.938

Means across three tasks. Midpoint: 1 step, 2 NFE; $\alpha = 0.005$, macro FPR $= .003$. Full metrics and variability: Appendix D.

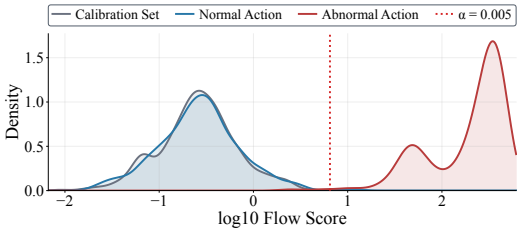


Figure 6: Reverse-flow score distributions for shift-perturbed Keypad action chunks using one reverse step, evaluated on 250 clean and 250 perturbed chunks.

Table 3: Task success rates (%), averaged across four stochastic wrappers.

Task	Unassisted	Residual	DDPM	CSA	GRIP	GRIP$^+$	LIFT	LIFT$^+$
Slalom	$40.6^{\pm 2.25}$	$38.4^{\pm 2.08}_{\blacktriangledown 2.2}$	$24.8^{\pm 1.50}_{\blacktriangledown 15.8}$	$49.6^{\pm 2.42}_{\blacktriangle 9.0}$	$41.9^{\pm 2.37}_{\blacktriangle 1.3}$	$40.7^{\pm 2.42}_{\blacktriangle 0.1}$	$58.2^{\pm 2.42}_{\blacktriangle 17.6}$	$\mathbf{59.2}^{\pm 2.42}_{\blacktriangle 18.6}$
CAN	$44.2^{\pm 2.58}$	$55.9^{\pm 2.30}_{\blacktriangle 11.7}$	$49.5^{\pm 2.19}_{\blacktriangle 5.3}$	$58.8^{\pm 2.58}_{\blacktriangle 14.6}$	$65.9^{\pm 2.27}_{\blacktriangle 21.7}$	$64.2^{\pm 2.27}_{\blacktriangle 20.0}$	$62.9^{\pm 2.58}_{\blacktriangle 18.7}$	$\mathbf{66.8}^{\pm 2.42}_{\blacktriangle 22.6}$
Keypad	$48.5^{\pm 1.00}$	$53.8^{\pm 0.92}_{\blacktriangle 5.3}$	$63.1^{\pm 1.08}_{\blacktriangle 14.6}$	$62.4^{\pm 0.92}_{\blacktriangle 13.9}$	$\mathbf{72.4}^{\pm 0.92}_{\blacktriangle 23.9}$	$\mathbf{72.4}^{\pm 0.87}_{\blacktriangle 23.9}$	$67.2^{\pm 1.00}_{\blacktriangle 18.7}$	$67.9^{\pm 1.00}_{\blacktriangle 19.4}$

Equal-weight means across four wrappers. Upper-right $\pm$ values are 95% bootstrap CI half-widths; all columns are recomputed from raw episodes using group-stratified resampling within each wrapper ($B = 10^4$). Lower-right colored triangles show changes from Unassisted in percentage points; $+$: OOD gating. Bold: best value per task.

6.3 What Does the Adapter Correct? A Controlled Perturbation Analysis

Section 6.2 reports end-to-end success rates, which do not show whether GLOVES modifies the actions that should be modified. The improvement could also come from recovery after an erroneous action has already been executed. Because agent rollouts do not indicate which actions are erroneous, we construct a controlled setting in which they are known.

Setup. To probe the effectiveness of our method, we construct a near-perfect agent by collecting goal-conditioned expert dataset and train a imitation learning (IL) agent (Appendix B.5.1). We further apply one of four stochastic perturbations to the IL agent's proposed action chunks. **Noised** adds Gaussian noise to every action dimension, **Laggy** repeats the previous chunk, **Slow** scales the chunk by a constant factor, and **Shift** adds a constant offset to one action dimension. Parameters are listed in Appendix Table 5. The four types of perturbation are widely used in current work (Reddy et al., 2018; Schaff & Walter, 2020; Yoneda et al., 2023; McMahan et al., 2024; Sun et al., 2025a; Kumar et al., 2021), which are expected to represent common failure cases in agents. We also compare against a residual correction baseline (Residual), which is given the perturbation types during training. It learns a residual r_ϕ that maps a perturbed agent action chunk back to the original one:

$$\mathcal{L}(\phi) = \mathbb{E}_{(I,a^e)\sim\mathcal{D}_e}\, \mathbb{E}_c \left\| r_\phi\big(I, c(a^e)\big) - \big(a^e - c(a^e)\big) \right\|_2^2, \tag{15}$$

where c is the identity or one of the four perturbation types (Appendix B.6). At test time, it corrects every proposed chunk as $\tilde{a} = a_u + r_\phi(I, a_u)$.

This construction lets us separately evaluate whether the gate detects perturbed proposals and whether the editors recover task performance from those perturbations.

One-step OOD scoring. For selective intervention, the gate should avoid altering valid actions while adding minimal inference cost, so we prioritize a low false-positive rate and efficient scoring. With one midpoint ODE reverse step, the score achieves a clean false-positive rate of 0.003 at $\alpha = 0.005$, with macro precision 0.975, recall 0.758, and AUROC 0.938 across the three tasks and four perturbation types. Scoring takes 0.356 ms per sample, compared with 0.625 ms for five steps, reducing scoring time by approximately 43%. We therefore use one step as the default gating setting; Appendix D reports the complete $1/5/50$-step comparison and paired toy visualizations.

The main reduction in wrapper-event recall comes from **Laggy**: under the balanced evaluation protocol, laggy actions account for approximately 69% of missed perturbation events, with a recall of 0.344 (Table 2). This reflects an important distinction between detecting a synthetic perturbation and

identifying an action that requires intervention. Because laggy repeats the previous action chunk, a repeated chunk can remain compatible with the current observation despite being labeled as a wrapper event. Wrapper-event recall therefore does not directly measure how often the gate misses actions that would cause execution failure.

Figure 6 provides a representative example on Keypad with the **Shift** perturbation. One-step midpoint scoring separates the perturbed and clean action chunks around the calibrated threshold, rejecting all perturbed chunks while flagging only a small fraction of clean chunks in this example.

Recovery across perturbation types. We next ask whether intervention can recover task performance from the controlled action perturbations. Averaged across the four perturbation types, both editors improve over the unassisted agent on every task (Table 3), showing that their gains persist after averaging over different failure modes.

We next examine the individual task–wrapper settings to determine whether these average gains are driven by only a subset of perturbations. Across the 12 settings reported here, ungated GRIP and LIFT each outperform the corresponding unassisted agent in 10 settings, and at least one GLOVES variant improves performance in all 12. The recovery pattern nevertheless differs across perturbation types. Averaged across tasks, GRIP is stronger on Noised and Shift perturbations, whereas LIFT is stronger on Laggy and Slow perturbations. The difference is particularly pronounced for Shift, where GRIP raises average success from 45.1% to 71.8%, while for Laggy, LIFT raises it from 46.9% to 57.9%. Both editors provide large gains under Slow perturbations.

These results show that the aggregate improvements are not driven by a single perturbation type: recovery is observed across all four controlled failure modes, although the magnitude of the gains varies across editors and perturbations.

Cost of selective intervention. Let r denote the intervention rate and N_{edit} the NFE of one correction. A one-step midpoint gate costs two evaluations:

$$N_{\text{gated}} = r(2 + N_{\text{edit}}) + (1 - r)2 = 2 + rN_{\text{edit}}, \qquad \Delta N = (1 - r)N_{\text{edit}} - 2. \tag{16}$$

The OOD gate enables selective correction at a measured scoring cost of 0.356 ms per sample. Its net computational benefit depends on the intervention rate and the cost of action correction.

6.4 CHOOSING BETWEEN GRIP AND LIFT

We recommend GRIP as the default when source-agent rollouts are unavailable or latency and memory are constrained. When agent rollouts and additional compute are available, LIFT provides an alternative that explicitly models the source action distribution. A small validation set can then determine whether its success gain justifies the additional cost. Appendix C.4 reports the parameter sensitivity of both editors.

7 CONCLUSION

We presented GLOVES, which improves the actions of a frozen agent without updating it. A goal-agnostic expert flow, learned from limited demonstrations, refines the agent's proposed action chunks, while the proposal itself supplies the task intent. On three simulation tasks, the better GLOVES editor outperforms every baseline for both a finetuned VLA and a code-as-policy agent, and on a real UR5 it raises the VLA's charger-insertion success from 45% to 100%. Controlled perturbations show that the flow's reverse score flags perturbed chunks with high precision, and that both editors recover performance across all four perturbation types, beyond a residual corrector trained on those perturbations. GRIP needs only the expert flow and is the stronger editor for magnitude and offset errors, whereas LIFT, which also models the agent's actions, better preserves intent when the goal-agnostic prior cannot infer it.

AI USE STATEMENT

We used generative AI tools to assist with literature discovery and summarization, translation and manuscript editing, and feedback on experimental methodology and interpretation of results. We also used AI assistance to write SLURM job-launch scripts, adapt baseline implementations, develop the CaP controllers, and generate plotting code. The core GLOVES algorithms were designed and implemented by the authors. We reviewed the AI-assisted references, code, analyses, and text, and take responsibility for the final content of this work, including all reported results and claims.

ETHICS STATEMENT

This work studies robotic action adaptation through simulation and laboratory experiments. Although GLOVES aims to improve execution reliability, its OOD gate measures conformity to an expert distribution and does not provide a safety guarantee. Deployment beyond the evaluated settings requires task-specific validation and appropriate robot safety measures.

REPRODUCIBILITY STATEMENT

The main text specifies the problem formulation, OOD scoring procedure, and action-editing updates. Algorithm 1 provides pseudocode for LIFT. Appendix B describes the experimental tasks, agent configurations, controlled perturbations, and residual baseline. Model architecture and training settings are provided in Appendix C, while Appendix D details the OOD evaluation and reverse-step comparisons. Appendix C.4 reports sensitivity to the editing parameters.

REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael 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, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

Max Braun, Noémie Jaquier, Leonel Rozo, and Tamim Asfour. Riemannian flow matching policy for robot motion learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5144–5151, 2024.

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. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

OX-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, et al. Open X-Embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

Ruiqi Feng, Chenglei Yu, Wenhao Deng, Peiyan Hu, and Tailin Wu. On the guidance of flow matching. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 16993–17029, 2025.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 9902–9915, 2022.

Joshua Jones, Oier Mees, Carmelo Sferrazza, Kyle Stachowicz, Pieter Abbeel, and Sergey Levine. Beyond sight: Finetuning generalist robot policies with heterogeneous sensors via language grounding. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5961–5968, 2025.

Chanhyuk Jung, Sangwon Kim, Kwang-Ju Kim, Dasom Ahn, Joonki Baek, Sungkeun Yoo, and Byoung Chul Ko. Flow-guided policies: Overcoming diffusion limitations for robust robot imitation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 2507–2512, 2025.

Jeongsol Kim, Yeobin Hong, Jonghyun Park, and Jong Chul Ye. FlowAlign: Trajectory-regularized, inversion-free flow-based image editing. *arXiv preprint arXiv:2505.23145*, 2025a.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. 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.

Vladimir Kulikov, Matan Kleiner, Inbar Huberman-Spiegelglas, and Tomer Michaeli. Flowedit: Inversion-free text-based editing using pre-trained flow models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19721–19730, 2025.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid motor adaptation for legged robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2021.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591.

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

Jason Jingzhou Liu, Yulong Li, Kenneth Shaw, Tony Tao, Ruslan Salakhutdinov, and Deepak Pathak. FACTR: Force-attending curriculum training for contact-rich policy learning. *arXiv preprint arXiv:2502.17432*, 2025.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Brandon J McMahan, Zhenghao Peng, Bolei Zhou, and Jonathan C Kao. Shared autonomy with ida: interventional diffusion assistance. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 128330–128354, 2024.

Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Vivek Myers, Bill Chunyuan Zheng, Oier Mees, Sergey Levine, and Kuan Fang. Policy adaptation via language optimization: Decomposing tasks for few-shot imitation. *arXiv preprint arXiv:2408.16228*, 2024.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Mingjie Pan, Siyuan Feng, Qinglin Zhang, Xinchen Li, Jianheng Song, Chendi Qu, Yi Wang, Chuankang Li, Ziyu Xiong, Zhi Chen, et al. SOP: A scalable online post-training system for vision-language-action models. *arXiv preprint arXiv:2601.03044*, 2026.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

Siddharth Reddy, Anca D Dragan, and Sergey Levine. Shared autonomy via deep reinforcement learning. *arXiv preprint arXiv:1802.01744*, 2018.

Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

Moritz Reuss, Hongyi Zhou, Marcel Ruehle, Omer Erdinc Yagmurlu, Fabian Otto, and Rudolf Lioutikov. FLOWER: Democratizing generalist robot policies with efficient vision-language-action flow policies. *arXiv preprint arXiv:2509.04996*, 2025.

Charles Schaff and Matthew R Walter. Residual policy learning for shared autonomy. *arXiv preprint arXiv:2004.05097*, 2020.

Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Lucy Xiaoyang Shi, Zheyuan Hu, Tony Z. Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. Yell at your robot: Improving on-the-fly from language corrections. *arXiv preprint arXiv:2403.12910*, 2024.

Luzhe Sun, Jingtian Ji, Xiangshan Tan, and Matthew Walter. FlashBack: Consistency model-accelerated shared autonomy. In *Proceedings of the Conference on Robot Learning (CoRL)*, pp. 924–940, 2025a.

Luzhe Sun, Takuma Yoneda, Samuel W Wheeler, Tianchong Jiang, and Matthew R Walter. StackGen: Generating stable structures from silhouettes via diffusion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6235–6242, 2025b.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. Scaling proprioceptive-visual learning with heterogeneous pre-trained transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 124420–124450, 2024.

Zhendong Wang, Max Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025.

Edgar Welte, Yitian Shi, Rosa Wolf, Maximillian Gilles, and Rania Rayyes. FlowCorrect: Efficient interactive correction of generative flow policies for robotic manipulation. *arXiv preprint arXiv:2602.22056*, 2026.

Charles Xu, Qiyang Li, Jianlan Luo, and Sergey Levine. RLDG: Robotic generalist policy distillation via reinforcement learning. *arXiv preprint arXiv:2412.09858*, 2024.

Chen Xu, Tony Khuong Nguyen, Emma Dixon, Christopher Rodriguez, Patrick Miller, Robert Lee, Paarth Shah, Rares Ambrus, Haruki Nishimura, and Masha Itkina. Can we detect failures without failure data? Uncertainty-aware runtime failure detection for imitation learning policies. *arXiv preprint arXiv:2503.08558*, 2025.

Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, and Dieter Fox. ManiFlow: A general robot manipulation policy via consistency flow training. *arXiv preprint arXiv:2509.01819*, 2025.

Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Chenyang Zhao, Piaopiao Jin, Guokang Sun, Shaoqing Xu, et al. FPC-VLA: A vision-language-action framework with a supervisor for failure prediction and correction. *Expert Systems with Applications*, 2026.

Takuma Yoneda, Luzhe Sun, Ge Yang, Bradly Stadie, and Matthew Walter. To the noise and back: Diffusion for shared autonomy. *arXiv preprint arXiv:2302.12244*, 2023.

Wenye Yu, Jun Lv, Zixi Ying, Yang Jin, Chuan Wen, and Cewu Lu. Armada: Autonomous online failure detection and human shared control empower scalable real-world deployment and adaptation. *arXiv preprint arXiv:2510.02298*, 2025.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

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.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. FlowPolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 14754–14762, 2025.

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.

A LEARNED INVERSION-FREE TRANSPORT (LIFT) DETAILS

A.1 LIFT PSEUDOCODE

Algorithm 1 LIFT: Learned Inversion-Free Transport under the $x_0 \to x_1$ Convention

Require: source action a_u, image context c, agent velocity field v_u, expert velocity field v_e
Require: steps K, time grid $0 = t_0 < \cdots < t_K = 1$, edit scale η_{edit}, alignment weight λ_{align}, CFG scale ω

$a^{(0)} \leftarrow a_u$
for $k = 0, \ldots, K-1$ **do**
  $\Delta t_k \leftarrow t_{k+1} - t_k$, $\epsilon_k \sim \mathcal{N}(0, I)$
  $q_k \leftarrow (1 - t_k)\epsilon_k + t_k a_u$
  $p_k \leftarrow q_k + \left(a^{(k)} - a_u\right)$
  $v_q \leftarrow v_u(q_k, t_k \mid c)$
  $v_p \leftarrow v_u(p_k, t_k \mid c) + \omega\,[v_e(p_k, t_k \mid c) - v_u(p_k, t_k \mid c)]$
  $\hat{a}_u \leftarrow q_k + (1 - t_k)v_q$, $\hat{a}_e \leftarrow p_k + (1 - t_k)v_p$
  $a^{(k+1)} \leftarrow a^{(k)} + \eta_{\text{edit}}\Delta t_k(v_p - v_q) + \lambda_{\text{align}}(\hat{a}_u - \hat{a}_e)$
end for
return $\tilde{a} \leftarrow a^{(K)}$

A.2 CONVENTION TRANSLATION.

Our implementation uses the $x_0 \to x_1$ convention, where $x_0 = \epsilon \sim \mathcal{N}(0, I)$ is noise and $x_1 = a$ is the clean action chunk:

$$x_t = (1 - t)\epsilon + ta, \qquad v(x_t, t) = a - \epsilon.$$

Some FlowAlign derivations use the opposite convention, where the clean sample is indexed as x_0 and noise as x_1. This is only a reparameterization of the same path. If $s = 1 - t$, then

$$x_t = (1-t)\epsilon + ta = (1-s)a + s\epsilon, \quad v_t = -v_s.$$

Therefore, the two conventions are equivalent as long as the velocity sign, endpoint estimate, and update step are translated consistently. In our convention, the clean endpoint estimate is

$$\hat{a} = x_t + (1 - t)v(x_t, t),$$

and the FlowAlign update uses a positive $\Delta t_k = t_{k+1} - t_k$. If the implementation uses $\sigma = 1 - t$, then $\Delta t_k = -\Delta\sigma_k$.

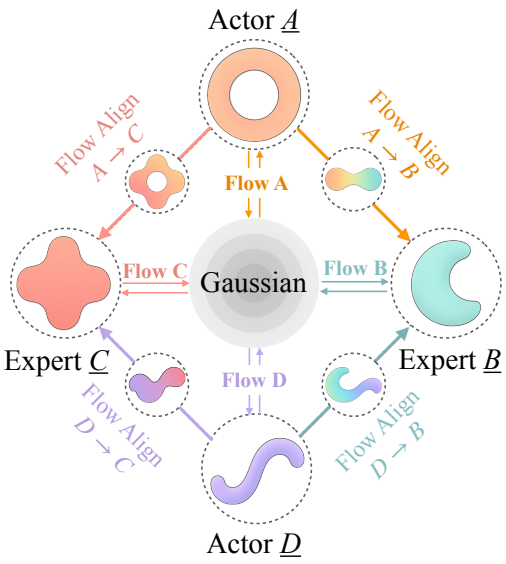


Figure 7: GLOVES provides transformations between imperfect agent and expert distributions.

B EXPERIMENTAL SETUP

B.1 SIMULATION ENVIRONMENTS

We evaluate GLOVES on three simulated robot-assistance benchmarks that jointly cover low-dimensional planar navigation, contact-rich manipulation and long-horizon sequential interaction. The three tasks were chosen so that the action-adaptation problem is exercised across qualitatively different action spaces (from 2-D Cartesian increments to 8-D joint-delta-plus-gripper commands), visual modalities (single- versus dual-view RGB, with or without proprioceptive state), and mode structures (small discrete sets of corridors and placement regions, large combinatorial sequence spaces, and continuously parameterized goal poses). All environments follow a Gym-style interface with a fixed per-episode budget; an episode terminates upon task completion, upon a task-specific failure condition (collision, wrong-button activation, or out-of-tolerance contact, as applicable), or upon reaching the maximum number of environment steps. Unless stated otherwise, RGB observations are rendered at a resolution of 256×256, action chunks have horizon $T = 10$, and an emitted

chunk is unrolled open-loop in the environment before the next policy query is issued. Table 4 summarizes the key dimensions of each task; the remainder of this section provides per-task descriptions of the environment, action interface, observation interface, mode/goal structure, success and failure criteria, and expert demonstration set.

Task	Simulator	Action space	Visual input	Modes / goals	Horiz.	Demos
Slalom	MuJoCo	2-D Δxy	1 side camera	24 corridor layouts	300	400
CAN	Robosuite	7-D delta EEF + gripper	side and wrist camera	4 placement regions	400	400
Keypad	MuJoCo	3-D Δxyz	side and wrist camera	$9^3 = 729$ button codes	1000	450

Table 4: Summary of the three simulation benchmarks used to evaluate GLOVES.

B.2 SLALOM

We adopt the obstacle-avoidance environment from the D3IL benchmark, implemented in MuJoCo, in which a planar manipulator must traverse a corridor populated by six pillars arranged between a start line and a goal line. The robot controls planar end-effector displacements via 2-D Δxy actions with fixed height and orientation. The environment exposes $24 = 2 \times 3 \times 4$ obstacle-layout modes. These modes act as a discrete categorical label that determines which corridor must be traversed for the episode to count as a success. Observations consist of a single third-person RGB image at 256×256 resolution. An episode terminates when the end-effector crosses the goal line in the prescribed corridor (success), when contact with any pillar is detected (collision), or when the per-episode budget of 300 environment steps is exhausted (timeout). The expert dataset contains 400 trajectories (100 per goal-region label), recorded as observation-action pairs with per-trajectory mode annotations.

For evaluation, we run 30 episodes for each random seed, and each model is evaluated over 10 random seeds.

B.3 ROBOMIMIC CAN

We evaluate the can placement task from RoboMimic, executed in the Robosuite simulator on a MuJoCo backend. A Franka Emika Panda arm must transport a soup can from a randomly initialized position into one of four discrete target placement regions. The four regions {top-left, top-right, bottom-left, bottom-right} are exposed as distinct task modes, providing a multimodal expert distribution rather than the single-mode protocol of the original benchmark. Actions are 7-dimensional delta end-effector pose, and a 1-D continuous gripper command (open at -1, closed at $+1$). Observations consist of two RGB streams: an external third-person camera and a wrist-mounted camera, both rendered at 256×256 resolution and updated at the 20 Hz control frequency. An episode terminates when the simulator reports a successful placement of the can within the requested region (success) or after 400 environment steps (timeout). The expert dataset comprises 400 trajectories (100 per region, totalling 60,399 transitions) collected from a region-conditioned scripted planner with mild target-pose noise ($\sigma_x = \sigma_y = 0.05\,\mathrm{m}$) to induce diversity around each placement target.

For evaluation, we run 32 episodes for each random seed, and each model is evaluated over 10 random seeds.

B.4 KEYPAD

We introduce a custom MuJoCo manipulation task in which the robot must enter a 3-button code on a 3×3 keypad in a prescribed order. Actions are 3-D Cartesian displacements; the low-level controller appends a constant downward-facing quaternion so that the end-effector consistently approaches the keypad face from above. The task admits $9^3 = 729$ distinct modes, each corresponding to a three-digit button sequence drawn uniformly per episode. Observations include both a keypad-facing RGB camera and a wrist-mounted RGB camera, each rendered at 256×256. An episode terminates when

the requested sequence is successfully entered (success), when an incorrect button is depressed at any sub-step (wrong-button failure), upon undesired contact with the keypad frame (collision), or after 1000 environment steps (timeout). The expert dataset contains 450 demonstrations (50 per single-button press across 9 buttons), each averaging approximately 66 control steps; multi-press episodes are obtained at training time by composing single-press demonstrations.

A distinguishing property of this task is that the language condition is *per-timestep* rather than per-trajectory: the "current target button" advances each time a sub-goal is completed within the same episode. The policy is therefore supplied with a one-hot encoding of the *current* target button only, while the image-conditioned assistive policy receives no language signal and must infer the active sub-goal from RGB observations alone.

For evaluation, we run 30 episodes for each random seed, and each model is evaluated over 10 random seeds.

B.5 Agent Policy

B.5.1 Imitation Learning Policy with wrapper

The first type of agent policy used in simulation is a goal-conditioned imitation learning policy. It shares the same architecture as the flow matching model, with the only difference being that the IL agent additionally takes the goal as input. We then apply stochastic perturbation wrappers to the agent policy in order to inject structured failure modes into its actions.

Formally, each wrapper transforms an action chunk $a \in \mathbb{R}^{B\times T\times A}$, where B is the number of parallel environments, $T = 10$ is the chunk horizon, and A is the action dimension. Randomness is sampled independently for each environment using decorrelated child generators derived from a shared `numpy.random.Generator`.

Noised Wrapper. The noised wrapper injects i.i.d. Gaussian perturbations into each action dimension:

$$a_i \leftarrow a_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I).$$

This models imprecise or jittery control behavior.

Laggy Wrapper. With probability p_{rep}, the laggy wrapper replays the previous action chunk instead of emitting a newly predicted chunk. This simulates delayed reactions, stalls, or persistent human corrections.

Slow Wrapper. With probability p_{slow}, the entire action chunk is scaled by a factor $\alpha < 1$:

$$a_i \leftarrow \alpha a_i.$$

This models hesitant or low-confidence behavior where the intended motion direction is correct but under-actuated.

Shift Wrapper. With probability p_{shift}, a constant offset α is added to one action dimension d across the full chunk. This introduces systematic control bias, such as calibration errors or directional drift.

Table 5: Stochastic-wrapper parameter configurations used for the three simulation tasks. Each cell specifies the parameter tuple for the named wrapper.

Task	**Noised** (σ)	**Laggy** (p_{rep})	**Slow** (α, p_{slow})	**Shift** (d, α, p_{shift})
Slalom	0.005	0.20	(0.5, 0.5)	(1, 0.005, 0.5)
Keypad	0.015	0.30	(2.0, 0.5)	(1, 0.005, 0.5)
RoboMimic CAN	0.15	0.20	(0.5, 1.0)	(1, 0.20, 0.5)

B.5.2 Vision Language Action Model

We finetune FLOWER for the CAN, Slalom, Charger Real, and Cup Real VLA-agent experiments using task-specific demonstrations. These runs use a latent-space Gaussian-blur curriculum with

a linear schedule, following FACTR (Liu et al., 2025). To strengthen instruction following at inference, we apply language dropout during finetuning and combine the language-conditioned and language-dropped velocity predictions using classifier-free guidance (Ho & Salimans, 2022; Nichol et al., 2021):

$$v_{\text{cfg}}(x_t, t, c, L) = v_\theta(x_t, t \mid c, \varnothing) + s_{\text{cfg}} \left[v_\theta(x_t, t \mid c, L) - v_\theta(x_t, t \mid c, \varnothing) \right],$$

where c denotes the visual observation, L the language instruction, $\varnothing$ the language-dropped condition, and s_{cfg} the guidance scale. The finetuning settings and available CFG inference settings are summarized in Table 6.

Setting	CAN	Slalom	Charger Real	Cup Real
Learning rate	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Batch / GPUs	$24 \times 2 \times 4 = 192$	$40 \times 1 \times 4 = 160$	$24 \times 2 \times 4 = 192$	$24 \times 2 \times 4 = 192$
Training length	50 epochs	50 epochs	50 epochs	50 epochs
Latent curriculum	Gaussian blur, linear schedule			
Language dropout	0.2			
CFG scale s_{cfg}	1.5	2.0	1.0	2.0

Table 6: FLOWER finetuning settings and CFG inference hyperparameters for the VLA agent experiments.

The exact training-time language instructions are:

- **CAN:** "Pick up the can and place it in the top left of the destination bin."; "Pick up the can and place it in the top right of the destination bin."; "Pick up the can and place it in the bottom left of the destination bin."; "Pick up the can and place it in the bottom right of the destination bin."
- **Slalom:** We use three space-separated mode integers as the instruction. The mode set is $\{0, 1\} \times \{0, 1, 2\} \times \{0, 1, 2, 3\}$, rendered as strings such as `0 0 0` through `1 2 3`.
- **Charger Real:** "Insert the charger plug into the left target socket."; "Insert the charger plug into the right target socket."
- **Cup Real:** "Put the cup on the top coaster."; "Put the cup on the middle coaster."; "Put the cup on the bottom coaster."

B.5.3 Code-as-Policy

We construct demonstration-guided code-as-policy (CaP) agents for CAN, Slalom, and Keypad, inspired by Code as Policies (Liang et al., 2023). A language-model coding agent develops task-specific Python controllers that receive goal instructions and predict action chunks from image observations. The CaP can develop and iterate with demonstration data and rollout feedback. This extends the idea of expressing robot policies as executable programs to our demonstration-guided setting. The evaluated CaPs and fitted parameters are frozen; execution requires no language-model calls or online code generation.

Each CaP uses ten demonstration trajectories containing RGB images, actions and corresponding language instructions. Generally, the coding agent would fit local mappings from actions to image-feature displacements using regularized least squares, and extract task-specific waypoints, interaction offsets, or motion profiles. Visual servoing uses these mappings to correct image-space errors. We allow at most ten rounds of rollout feedback for CaP to optimize per task. No neural action policy is trained. Table 7 summarizes the common settings.

At runtime, CaP uses the frozen SAM ViT-B segmentation, prompted by deterministic image-based heuristics and combined with geometric image processing, to extract object and tool locations. CAN and Keypad additionally use a wrist view; Keypad uses detected button radii as a visual cue for approach and retraction. CaP maintains internal phase and execution history.

The task-specific CaPs operate as follows:

- **CAN:** The instruction specifies one of four destination-bin regions. Main- and wrist-view features guide a phase-based controller through approach, grasp, lift, transport, descent,

Setting	CAN	Slalom	Keypad
Demonstrations	10	10	10
RGB views used	Main + wrist	Main	Main + wrist
Image processing size		256×256 per view	
Action dimension	7	2	3
Prediction / execution		$H = 10, K = 10$	
Runtime language-model calls		None	

Table 7: Construction and execution settings for the CaP agents. Action dimensions follow each environment's existing control interface.

and release. Demonstrations provide contact and transport offsets, while visual feedback governs alignment and phase progression.

- **Slalom:** The instruction is a route tuple in $\{0,1\} \times \{0,1,2\} \times \{0,1,2,3\}$. Image-detected obstacles define route-dependent waypoints. A fitted visual servo generates planar actions, and observed tool position determines progress through the route.
- **Keypad:** The instruction specifies an ordered button sequence. A demonstration-derived grid and visual servo guide the tool to each button. Demonstration-derived vertical press and retract profiles are combined with visual contact and clearance checks. Phase bookkeeping is synchronized with the actions actually issued, including any truncated final chunk, before advancing to the next button.

If required visual features are unavailable, the CaPs attempt reacquisition and issue bounded recovery chunks, preserving the commanded gripper state where applicable. Persistent loss of required visual evidence beyond the 30-step recovery allowance terminates the rollout as a perception failure.

B.6 Residual Correction Baseline

No paired agent and expert actions are available, so the residual network r_ϕ in Equation 15 is trained on perturbations of the expert's own chunks $a \in \mathbb{R}^{T \times d_a}$. For each training sample, one operator c is drawn: the identity with probability 0.25, which gives a zero target, and otherwise one of four perturbation families chosen uniformly at random. The families are never composed:

$$
\begin{aligned}
&c_{\text{noise}}(a) = a + \varepsilon, && \varepsilon \sim \mathcal{N}(0, \sigma^2 I), && \sigma \sim \text{LogU}[0.001,\ 1.0],\\
&c_{\text{scale}}(a) = \alpha\, a, && && \alpha \sim \text{LogU}[0.01,\ 2.0],\\
&c_{\text{shift}}(a) = a + s\,\beta\,\mathbf{1}_T e_d^\top, && d \sim \mathcal{U}\{1,\dots,d_a\},\ s \sim \mathcal{U}\{\pm 1\}, && \beta \sim \text{LogU}[0.01,\ 2.0],\\
&c_{\text{lag}}(a)_t = a_{t-m\Delta}, && m \sim \mathcal{U}\{1,\dots,5\}, &&
\end{aligned}
$$

where LogU denotes a log-uniform distribution and Δ is the number of steps executed per chunk; the lagged chunk is taken from the same demonstration. These ranges are identical for all tasks and are not set to the evaluation parameters in Table 5.

The residual network has the same architecture and parameter count as the task's expert flow, with a constant time input so that r_ϕ is a deterministic map. We train it with batch size 256, learning rate 3×10^{-4}, weight decay 10^{-3}, and 50 epochs, hold out 10% of the trajectories for validation, and use the best validation checkpoint. At test time, the corrected chunk $a_u + r_\phi(I, a_u)$ is clipped to the action bounds and executed at every query, without OOD gating. Evaluation uses the same perturbations, seeds, and episode counts as the other methods.

B.7 Real World Experiment

Task	Action space	Visual input	Targets	Demos	Transitions
UR5 Charger	3-D Cartesian setpoint	side + wrist camera	2 sockets	200	22,797
UR5 Cup	7-D end-effector pose	side + wrist camera	3 cups	235	45,409

Table 8: Summary of the two real-world UR5 tasks used to evaluate GLOVES.

UR5 Charger Insertion The robot must align and insert a charger into one of two sockets under visual feedback from one side camera and one wrist camera. This task stresses fine pose correction near contact, where small translation errors translate to missed insertions. The dataset contains 200 successful scripted trajectories (100 per socket) recorded at 10 Hz, totaling 22,797 control steps.

Evaluation criterion. A trial is considered successful if the agent inserts the charger into the target socket specified by the language instruction within 160 environment steps. Failures include unsuccessful insertion, misalignment, or excessive contact forces that trigger an emergency stop. At the beginning of each episode, the charger's pose is reset by grasping it from a fixed initial location. For each experimental setting, we evaluate 40 episodes in total (20 episodes per target socket) and report the success rate.

UR5 Fluid Carrying The task requires the robot to transport a cup to one of three target locations while avoiding a fixed obstacle. The dataset contains 235 successful demonstrations, approximately balanced across the three targets, totaling 45,409 control steps.

Evaluation criterion. A trial is successful if the agent places the cup at the target pose specified by the language instruction within 160 environment steps. Failure cases include a positional error greater than 0.025 m in the x or y direction, a positional error greater than 0.015 m in the z direction, or any collision with the obstacle.

C Model Architecture

We instantiate the assistive policy with a Diffusion Transformer (DiT) velocity-field architecture (Sec. C.1) coupled to a shared visual and state encoder stack (Sec. C.2). Table 9 summarizes the architecture hyperparameters and training details.

C.1 DiT Velocity Field

The DiT velocity field is a Transformer that operates on a token set composed of (i) the chunked action tokens of the noisy sample x_t and (ii) a sequence of context tokens generated by the visual and state encoder stack. Each block consists of three pre-LayerNorm sub-layers: self-attention over action tokens with rotary positional embeddings applied to queries and keys; cross-attention from action queries to context keys and values without positional encoding on the context; and a GELU-activated MLP with expansion ratio $r \in \{2, 4\}$.

Each sub-layer is conditioned via adaptive layer normalization with a zero-initialized residual gate (AdaLN-Zero): the time embedding is mapped by a zero-initialized linear projection to per-layer scale and gate parameters. The time embedding itself is sinusoidal with frequency dimension 256 and is fed through a two-layer MLP to the hidden dimension; we scale t by 1000 before the sinusoidal embedding, in line with prevailing diffusion- and flow-matching practice.

C.2 Visual and State Encoders

We adopt an ImageNet-pretrained ResNet-18 with frozen batch normalization as the visual backbone in all main experiments. Features are extracted from the third residual stage (channel dimension 256, spatial stride 16); for a 256×256 input this yields a 16×16 token grid, i.e., 256 spatial tokens per view. A 1×1 convolution then projects the feature channels to the context dimension d_c. The backbone is fine-tuned end-to-end.

When multiple views are present, we instantiate *independent (non-weight-shared) encoders* per view, and the per-view token grids are concatenated along the sequence axis before being passed to the DiT cross-attention.

The proprioceptive state encoder is a three-layer feedforward network from input dimension S to width 256 to width 256 to output dimension d_c, and it produces a single context token consumed alongside the visual tokens.

C.3 Training Procedure

Table 9 consolidates the model and training hyperparameters used across all reported experiments. Both the standard flow-matching model and the FlowAlign model share the optimizer, and regularization settings in (b); their architectural sizes are listed separately in (a).

(a) Model hyperparameters

Component	Default
Visual backbone	ResNet-18
Pretrained weights	ImageNet
Feature stage	3rd residual
Feature channels	256
Spatial tokens per view	16×16
Image normalization	ImageNet stats
State encoder hidden width	256
FM context dim d_c	256
FM hidden width d	128
FM MLP ratio r	4
FlowAlign context dim d_c	512
FlowAlign hidden width d	512
FlowAlign MLP ratio r	2
Attention heads H	4
Block dropout	0.1
Time embedding dim	256
Time scaling factor	1000
Visual dropout p_{vd}	0.1
Action chunk length T	10

(b) Training hyperparameters

Component	Default
Optimizer	AdamW
Base learning rate	3×10^{-4}
Visual learning rate	3×10^{-5}
Visual LR scale	$0.1\times$
Weight decay	10^{-3}
Adam (β_1, β_2)	$(0.9,\ 0.999)$
Batch size	256
Training epochs	500
CFG dropout (training) p_{cfg}	0.2
Time-sampling schedule	Uniform
Karras p_{std}	1.2

Table 9: Default model (a) and training (b) hyperparameters used across all tasks.

C.4 Parameter Sensitivity

Lift. The screening sweep shows limited marginal sensitivity within evaluated ranges of all three editing parameters. Mean success rates vary by 1.10, 1.98, and 0.17 percentage points over $\omega \in \{0.2, 0.5\}$, $\eta_{\text{edit}} \in \{0.3, 0.5, 0.7\}$, and $\lambda_{\text{align}} \in \{0.05, 0.07, 0.10\}$, respectively, averaging over task–wrapper settings and remaining parameters.

Grip. Grip fixes $\lambda_{\text{align}} = 1$ and selects t_{init} for each setting. Table 10 isolates the energy term by comparing $\lambda_{\text{align}} = 0$ with $\lambda_{\text{align}} = 1$ at the same t_{init}. The energy term improves success in 9 of the 12 settings; in the remaining three it changes success by at most 2.0 percentage points. The largest gain is on Keypad with the Noised perturbation (+20.7).

Table 10: Ablation of the guidance strength for Grip.

Task	Wrapper	Baseline	$\lambda = 0$	$\lambda = 1$	Δ
Slalom	noised	40.0 ± 2.9	45.3 ± 2.9	46.7 ± 5.8	+1.3
	laggy	38.7 ± 2.8	33.7 ± 2.7	31.7 ± 3.8	−2.0
	slow	55.7 ± 2.9	49.3 ± 2.9	49.3 ± 5.6	0.0
	shift	28.0 ± 2.6	34.3 ± 2.8	40.0 ± 3.5	+5.7
Keypad	noised	34.7 ± 2.7	30.7 ± 2.7	51.3 ± 7.8	+20.7
	laggy	54.0 ± 2.9	57.7 ± 2.9	61.3 ± 6.3	+3.7
	slow	52.7 ± 2.9	93.3 ± 1.4	95.0 ± 2.6	+1.7
	shift	52.7 ± 2.9	81.0 ± 2.3	82.0 ± 3.5	+1.0
CAN	noised	50.9 ± 2.8	52.8 ± 2.8	52.5 ± 5.7	−0.3
	laggy	48.1 ± 2.8	51.2 ± 2.8	55.0 ± 6.3	+3.8
	slow	23.1 ± 2.3	58.1 ± 2.8	62.8 ± 3.5	+4.7
	shift	54.7 ± 2.8	92.8 ± 1.4	93.4 ± 3.3	+0.6
Mean across 12 cells		42.9	55.5	58.9	+3.5

C.5 Number of Expert Demonstrations

We retrain the expert flow of GRIP on nested subsets of 20, 50, 100, and 200 demonstrations drawn from the same dataset. All other settings follow the full-data model: the validation split and normalization statistics are shared, each model trains for 500 epochs, and we use the best validation checkpoint. Evaluation reuses the settings of the reported runs, including t_{init}, so only the flow checkpoint changes. Table 11 reports success averaged over the four perturbation types. With 50 demonstrations, GRIP reaches about 60% of its full-data gain over the unassisted agent, and with 200 demonstrations about 80%. CAN depends most on the amount of data. On Slalom, where the full-data gain is within noise, success shows no consistent trend with the number of demonstrations.

Table 11: Success rate (%) of GRIP with expert flows trained on different numbers of demonstrations, averaged over the four perturbation types. Full is the reported model trained on all demonstrations (Table 4).

Task	Unassisted	20	50	100	200	Full
Slalom	40.6	43.3	42.5	40.8	43.6	41.9
CAN	44.2	45.8	53.4	54.3	59.1	65.9
Keypad	48.5	60.1	65.0	65.1	69.0	72.4
Average	44.4	49.7	53.6	53.4	57.2	60.1

D Out of Distribution Detection and Toy Example

Conditional Reverse-Flow Score We evaluate whether the conditional reverse-flow score used by our OOD detector produces meaningful conformal acceptance regions in controlled two-dimensional settings. Let $v_\theta(x_t, t \mid c)$ be a conditional rectified flow trained to transport samples from a standard Gaussian prior $\mathcal{P}_Z = \mathcal{N}(0, I)$ to the class-conditional data distribution $\mathcal{P}_X(x \mid c)$. Given a test point x and condition c, we run the learned flow backward for N steps to obtain an approximate reversed latent $\hat{z}_N$ and non-conformity score s_N

$$\hat{z}_N(x; c) = \mathrm{ODE}_N^{-1}[v_\theta, x, 1 \to 0; c], \qquad s_N(x; c) = \|\hat{z}_N(x; c)\|_2^2,$$

where N denotes the number of reverse integration steps. The robotic experiments use the image context I; the toy examples use a class condition c. For each integration budget, calibration and test examples are scored with the same numerical procedure. Given n calibration scores $s_{N,i}$, we compute

$$p_N(x; c) = \frac{1 + \sum_{i=1}^{n} \mathbf{1}\{s_{N,i} \geq s_N(x; c)\}}{n + 1}.$$

Low scores indicate that the point maps to a typical region of the Gaussian prior under the condition-dependent reverse map, while high scores indicate a conditionally unlikely sample. The gate flags a proposal when $p_N \leq \alpha$. Calibration scores are computed before inference; only the test score and its rank are needed online. We use one midpoint step as the efficiency-oriented default and compare it with five and fifty steps below.

D.1 Detection on Controlled Action Perturbations

We evaluate the four implemented wrappers (*noised*, *laggy*, *slow*, and *shift*) on Slalom, CAN, and Keypad with 3 different reverse steps 1/5/50. Each task–wrapper–step setting contains 250 clean and 250 perturbed test chunks. Each task–step setting uses 500 expert calibration examples, with $\alpha = 0.005$. The evaluation uses the trained DiT with EMA weights. Reported aggregate metrics are equal-weight means over the 12 task–wrapper settings; a positive label records a wrapper activation, rather than a verified action failure.

One-step scoring achieves high precision and a low clean false-positive rate while reducing scoring time by approximately 43% relative to five steps (Table 12). Five steps improve wrapper-event recall

Table 12: Reverse-step sensitivity with midpoint integration. Metrics are macro-averaged over the 12 task–wrapper settings. Time is the benchmarked scoring time per sample, excluding action editing.

Steps	NFE	Precision	Recall	F1	FPR	AUROC	AUPRC	Time (ms)
1	2	0.975	0.758	0.805	0.003	0.938	0.948	0.356
5	10	0.989	0.847	0.884	0.005	0.957	0.966	0.625
50	100	0.989	0.818	0.852	0.001	0.954	0.962	3.359

Table 13: One-step midpoint detection by wrapper. Each entry is the mean ± standard deviation across the three tasks.

Wrapper	Precision	Recall	F1	AUROC	AUPRC
Noised	0.997 ± 0.002	1.000 ± 0.000	0.999 ± 0.001	1.000 ± 0.000	1.000 ± 0.000
Laggy	0.913 ± 0.142	0.344 ± 0.305	0.454 ± 0.383	0.780 ± 0.042	0.820 ± 0.068
Slow	0.997 ± 0.003	0.931 ± 0.120	0.960 ± 0.068	0.988 ± 0.021	0.992 ± 0.014
Shift	0.994 ± 0.007	0.756 ± 0.416	0.808 ± 0.327	0.983 ± 0.030	0.980 ± 0.034

and ranking metrics; increasing the budget to fifty steps leaves macro AUROC at 0.954 while increasing scoring time to 3.359 ms. These measurements characterize the cost and detection behavior of each numerical score.

Laggy accounts for approximately 66% of missed wrapper events under one-step scoring (Table 13). This wrapper repeats a preceding expert action chunk, which can remain compatible with the current observation during continuous motion. **Consequently, wrapper-event recall is not the fraction of unsafe actions detected.** The lower laggy recall is consistent with this distinction, although these labels do not establish whether individual missed chunks are harmless. Downstream rollout success provides a separate assessment of the combined gate and editor.

Toy Distributions We consider an 8-class checkerboard distribution on $[-2, 2]^2$. The 4×4 grid contains data only in alternating cells, yielding eight occupied cells and eight empty cells (Figure 8).

Each occupied cell is assigned one class label, and the conditional flow receives the normalized class label $c = k/7$. This distribution is useful because a correct conditional detector should accept points inside the queried class cell and reject both empty cells and points from other class cells when they are evaluated under the wrong condition.

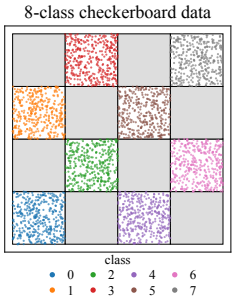


Figure 8: Eight-class checkerboard distribution used for conditional OOD evaluation.

Results The accepted-region plots (Figure 9) show that the learned reverse-flow score produces class-specific conformal acceptance regions. For each query label $c = k/7$, the accepted region $p(x, c) > \alpha$ concentrates in the corresponding occupied checkerboard cell, while empty cells and other class cells are rejected. This supports the use of the conditional reverse-flow prior score as a lightweight OOD detector.

We next test partial calibration coverage by fixing the query label to class 7 and progressively removing classes from the calibration set (Figure 10). When class 7 is included in calibration, the acceptance rate inside the class-7 cell is high. When class 7 is absent, performance persists, because the conditional reverse-flow scores remain comparable across classes.

Figure 11 evaluates points from classes 6, 5, and 0 under the incorrect class-7 condition, with calibration limited to classes 0–6. Their scores shift away from the reference distributions, and all three displayed wrong-label groups have 0.0% acceptance under both integration budgets. Together, the paired figures illustrate conditional discrimination with the one-step score and its five-step counter-

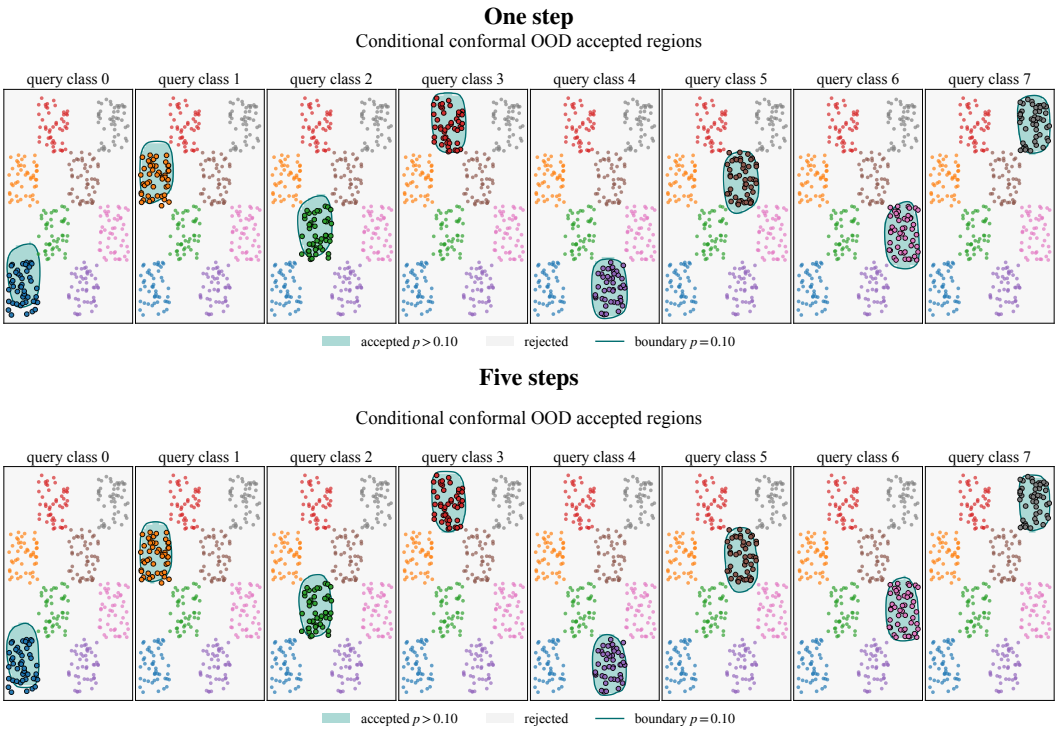


Figure 9: Conditional acceptance regions at $\alpha = 0.10$ using one reverse step (top) and five steps (bottom). Shaded regions satisfy $p_N(x; c) > \alpha$; contours mark the acceptance boundary. Each panel uses the displayed class as the query condition.

part in this controlled setting. Both integration budgets reject all displayed wrong-label groups and produce qualitatively similar class-specific acceptance regions.

D.2 Auxiliary validation through score-guided local search.

We further evaluate the effectiveness of the OOD score through an auxiliary score-guided local search experiment. Specifically, we test whether using the score to guide local changes to action chunks from the agent's distribution improves downstream task success (Table 14).[1]

As illustrated in Figure 12, we initialize the search at the agent's proposal, $a^{(0)} = a_u$, and evaluate candidates using

$$\ell_\tau(a; I, a_u) = \max(s(a; I) - \tau, 0) + \lambda_{\text{align}} \|a - a_u\|_2^2, \quad (17)$$

where τ is the calibrated score threshold corresponding to the gating level α. The loss penalizes scores above τ and large deviations ($\lambda_{\text{align}} \geq 0$) from the original proposal.

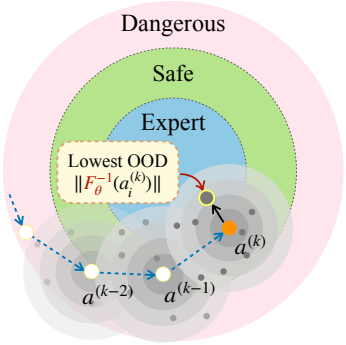


Figure 12: Score-guided local search.

At each iteration $k = 0, \ldots, K-1$, we draw M local perturbations and include a zero perturbation:

$$\epsilon_i^{(k)} \sim \mathcal{N}(0, \sigma^2 I), \quad i = 1, \ldots, M, \qquad \epsilon_0^{(k)} = 0. \quad (18)$$

The candidate chunks are

$$a_i^{(k)} = a^{(k)} + \epsilon_i^{(k)}, \qquad i = 0, \ldots, M. \quad (19)$$

[1] A low score suggests consistency with expert data but does not necessarily imply that an action is useful for the current task. This search procedure is used to evaluate the score and is not a component of the GLOVES assistance pipeline.

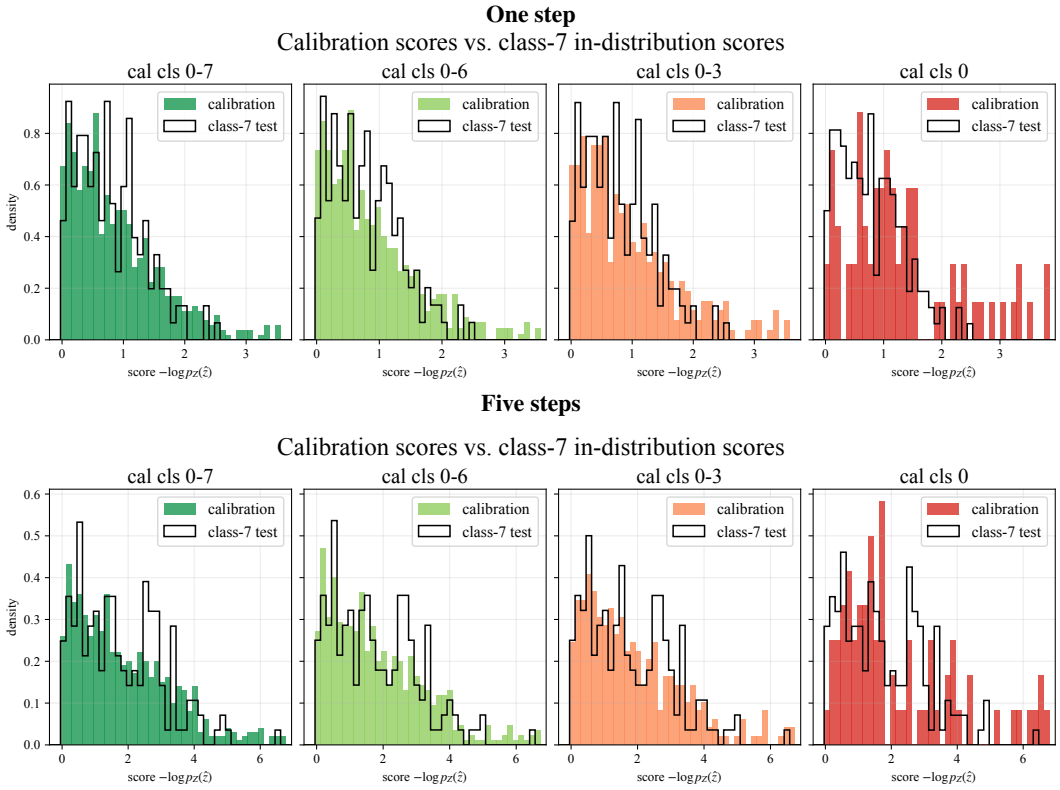


Figure 10: Calibration and true class-7 score distributions under reduced calibration coverage, using one reverse midpoint step (top) and five steps (bottom). Class-7 test scores overlap the calibration distributions in the displayed coverage settings.

The zero perturbation retains the current chunk as a candidate. We evaluate $\ell_i^{(k)} = \ell_\tau(a_i^{(k)}; I, a_u)$ and assign larger weights to candidates with smaller losses:

$$w_i^{(k)} = \frac{\exp\left[-\left(\ell_i^{(k)} - \ell_{\min}^{(k)}\right)\right]}{\sum_{j=0}^{M} \exp\left[-\left(\ell_j^{(k)} - \ell_{\min}^{(k)}\right)\right]}, \qquad \ell_{\min}^{(k)} = \min_{0 \le j \le M} \ell_j^{(k)}. \tag{20}$$

Subtracting the minimum loss provides numerical stability without changing the normalized weights. We then update the current chunk using the weighted perturbation:

$$a^{(k+1)} = a^{(k)} + \eta_{\text{edit}} \sum_{i=0}^{M} w_i^{(k)} \epsilon_i^{(k)}, \tag{21}$$

where $\eta_{\text{edit}} > 0$ controls the update magnitude. The regularization remains anchored at the original proposal a_u throughout the search. After K iterations, we execute the refined chunk $\tilde{a} = a^{(K)}$.

Setup. We run the search on the goal-conditioned IL agent of Section 6.3, under the four perturbations Noised, Laggy, Slow, and Shift with the parameters in Appendix Table 5. Evaluation uses the same episodes as the other methods in that section, so the Unassisted column matches Table 3. The search uses only the expert flow and the agent's proposal. It needs no goal, no task reward, and no additional training. For each task, we average success equally over the four perturbations.

Results. Table 14 shows that the search improves average success on all three tasks. The gain is largest on Keypad, from 48.5% to 65.0%, and smallest on CAN, from 44.2% to 46.5%. The search changes a chunk only to lower its score while keeping it close to the proposal, so these gains indicate that the score gives a useful local direction for correcting perturbed actions. The search is also expensive. It scores every candidate at each iteration and takes 1.68 s per query, about 22 times the FLOPs of a FLOWER query (Table 15). We therefore use it only to validate the score, not as an editor in GLOVES.

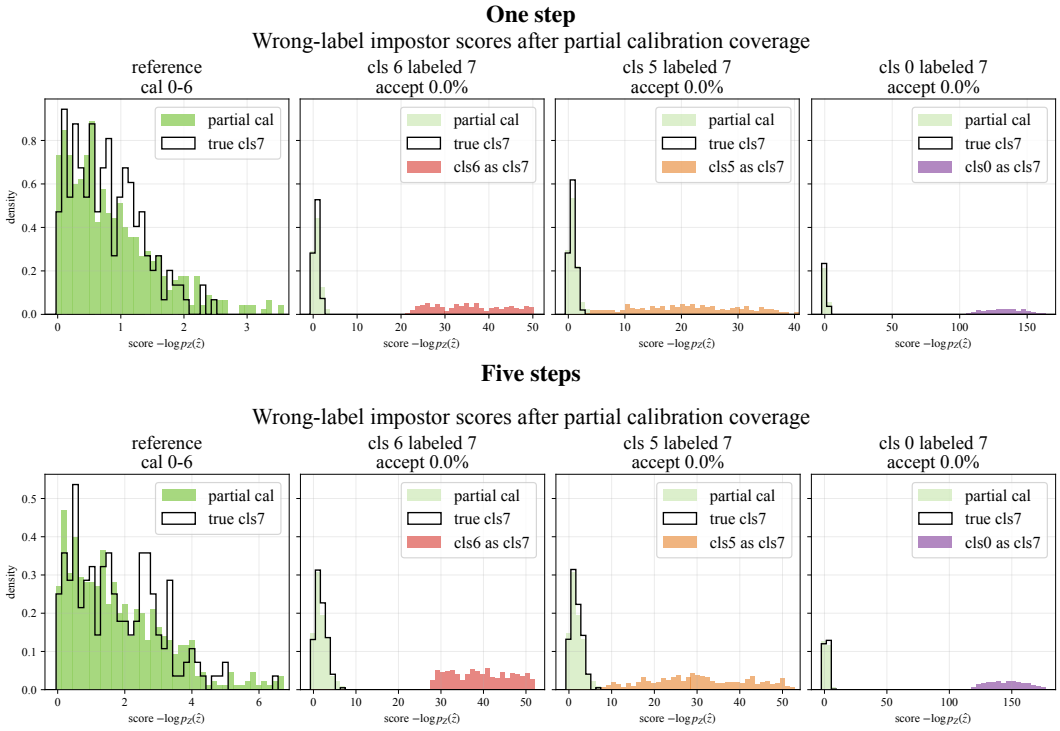


Figure 11: Mismatched class conditions with one reverse midpoint step (top) and five steps (bottom). Calibration uses classes 0–6; samples from classes 6, 5, and 0 are queried as class 7. Each displayed wrong-label group has 0.0% acceptance at $\alpha = 0.10$.

Table 14: Success rate (%) of the perturbed IL agent with and without score-guided local search, averaged over the four perturbations.

Task	Unassisted	Local search	Δ
Slalom	40.6	48.7 ± 3.63	$+8.1$
CAN	44.2	46.5 ± 2.70	$+2.3$
Keypad	48.5	65.0 ± 2.38	$+16.5$

E Computational Cost

Measurement setup. Table 15 profiles one adapter query $(I, a_u) \mapsto \tilde{a}$ at batch size one on an NVIDIA RTX A6000 with 16 AMD CPU cores. A query takes the image observation and the agent's proposed chunk and returns the refined chunk. GFLOPs/query counts both the conditioning and the velocity evaluations. Peak memory is the largest memory use during one query. VLA-query equiv divides GFLOPs/query by the cost of one FLOWER query, 888.15 GFLOPs. Median latency is measured per task, and the table reports the summary of these task-wise medians.

Call counts. Calls/query counts image encodings plus velocity-field evaluations; a batched call counts once. GRIP is profiled at $t_{\text{init}} = 0.8$ on a 100-step grid, so it integrates 20 intervals. Each interval is a predictor–corrector step with three velocity evaluations, which gives 60 velocity evaluations and 2 image encodings. The energy gradient acts only on the alignment term and adds no network calls. LIFT runs $K = 50$ editing steps and evaluates both flows at each step, which gives 100 velocity evaluations and 4 image encodings. Local search runs $K = 15$ iterations. Each iteration encodes the image once and scores all $M + 1 = 257$ candidates with one midpoint reverse step in a single batch, which gives 15 encodings and 30 velocity evaluations.

Results. GRIP costs 2.5% of a FLOWER query in FLOPs and takes 153.60 ms. LIFT costs 17.8% and takes 202.11 ms. LIFT is more expensive because it evaluates both the agent flow and the

expert flow at each editing step, while GRIP uses only the expert flow and integrates from t_{init} to 1. One action-chunk correction therefore takes about 0.15–0.20 s in this benchmark. The cost of GRIP grows with the number of integration intervals: on Slalom, its latency ranges from 34 ms at $t_{\text{init}} = 0.95$ to 323 ms at $t_{\text{init}} = 0.5$. Score-guided local search takes 1.68 s per query and 2192% of a FLOWER query in FLOPs, because it scores $M + 1$ candidates at each of its K iterations. We use it only to test whether the OOD score is a useful signal (Appendix D.2). It is not part of GLOVES.

Table 15: Batch-one adapter inference on an RTX A6000. Latency summarizes task-wise medians; FLOPs include conditioning and velocity evaluations. VLA-query equiv is relative to one FLOWER query (888.15 GFLOPs).

Method	Calls/query	GFLOPs/query	Peak memory	VLA-query equiv	Median latency (ms)
GRIP	62	22.1	43 MiB	2.5%	153.60
LIFT	104	157.9	527 MiB	17.8%	202.11
Local search (aux.)	45	19,466	2,768 MiB	2192%	1,682.8