DYNAMISM OF A TRAJECTORY: ENCODING COARSE INTENT INTO THE SOURCE OF FLOW-MATCHING POLICIES

Anonymous authors
Paper under double-blind review

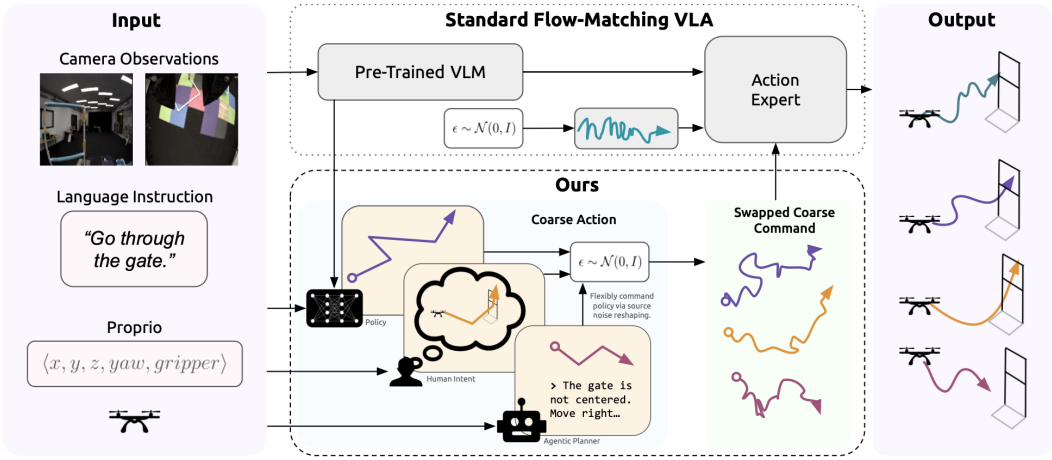


Figure 1: **Overview of our approach.** Our method modifies a pretrained flow-matching VLA through a coarse-to-fine manipulation of the sampled Gaussian noise, ϵ, used by the flow matching action expert. We first predict coordinates in coarse motion space (blue), which represents net motion over time. We then replace the coarse component of ϵ with the predicted one (green), leaving the action expert to denoise the residual, fine-grained motion. Because the coarse motion space is physically meaningful, a learned head, a person, or an agent can propose coarse actions, enabling it to more effectively produce trajectories outside of its training distributions.

ABSTRACT

Robot policies are often trained on a relatively small corpus of demonstrations of a few tasks. In such low-data regimes, generative policies do not need to learn how their embodiments move because reproducing demonstrated trajectories is enough to fit the data. As a result, such policies tend to be competent only along those routes. Therefore, to make best use of limited demonstrations, we separate coarse and fine motion by first fixing an orthonormal basis for the coarse part of an action chunk where the coordinates of the basis represent displacement over time, which we call the *command*. Then, we modify the initially drawn Gaussian noise by replacing its component along the coarse motion basis with the command. The command is constructed to be unchanged along the training path and absent from the regression target, leaving the flow to learn the residual, fine motion. While a learned head predicts the command by default, an external planner or human operator can override it, and a scalar input allows inference-time modulation of command adherence. We evaluate our method on a quadrotor platform on gate navigation and out-of-distribution maneuvers. On the demonstrated tasks, our policy improves success in simulation from 45% to 73%, where the images differ from those seen in training. Our method also enables flexible out-of-distribution command steering for drone maneuvers through and around gates, such as orbits and figure-eights, in simulation and on hardware. Finally, we show a preliminary demonstration in which an agent pilots the policy by approving or overriding its proposed commands.

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.

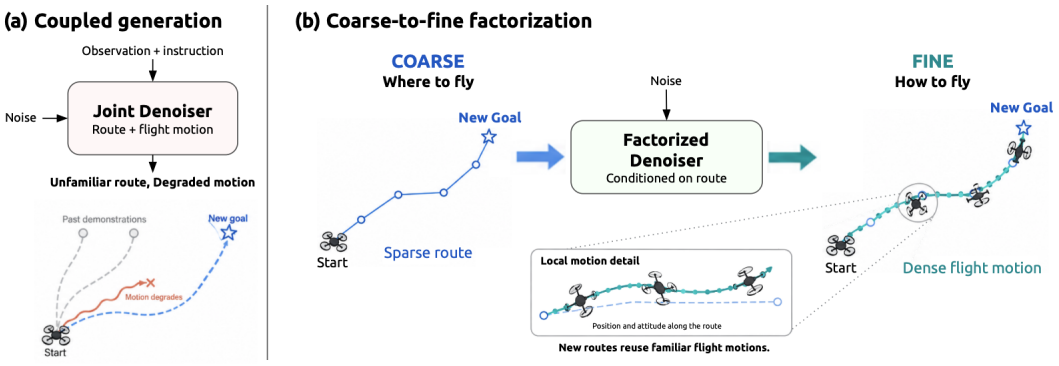


Figure 2: **Coupled generation vs. coarse-to-fine factorization. (a)** A typical denoiser generates the route and the flight motion together from the observation and instruction. When given a new goal outside the demonstrated routes, both the route and the motion quality degrade. **(b)** We factorize generation into two stages. The *coarse* stage decides *where to fly* by predicting a sparse route of waypoints to the new goal. The *fine* stage decides *how to fly*: a motion denoiser conditioned on that route generates dense flight motion (position and attitude). Because local motion is conditioned only on the route, familiar flight motion can be reused along new routes.

1 Introduction

Flow matching has become the standard generative backbone for vision-language-action (VLA) policies (Black et al., 2025; Bjorck et al., 2025). However, endowing VLAs with zero-shot capabilities remains relatively challenging (Zhou et al., 2025; Kim et al., 2025; Zhou et al., 2026). When data is limited to a small set of demonstrations across a few tasks (e.g., when simulation is unavailable or teleoperation is expensive), the typical approach is to perform supervised fine-tuning (Ghosh et al., 2024; Kim et al., 2024; Liu et al., 2025). In such settings, the training objective provides little incentive for a generative policy to learn motion strategies that generalize across scenarios: with limited data, simply memorizing the training trajectories in a lookup-table-like manner may suffice to minimize the training loss, as discussed in (He et al., 2026; Huang et al., 2026). The result is a competent policy, but only along the demonstrated routes.

We argue that this behavior arises because the current training paradigm must derive both coarse trajectory (where to go) and finer movement (how the embodiment realizes the trajectory) simultaneously during denoising; this entanglement means that the action head cannot succeed at one without the other. For instance, when an unfamiliar destination requires a familiar movement, it is likely that the whole output will degrade, even if the destination is eventually supplied, making generalization impossible (Figure 2a). Therefore, we propose that separating these two capabilities by isolating only finer movement to the denoiser would allow it to understand *how to move* even if the policy does not know where to move (Figure 2b). A natural question follows: how do we reliably train a model to learn these distinct capabilities?

Prior work demonstrates that the source distribution of a flow or diffusion model serves as a rich design surface. Learned source distributions accelerate and improve text-to-image generation (Kim et al., 2026), while refined initial noise can replace classifier-free guidance entirely (Ahn et al., 2026). We propose utilizing this design surface to support coarse, low-dimensional intent of where to go, leaving the high-frequency residual detail to the denoising process. To do this, we fix a small set of directions in action-chunk space that capture coarse motion. Projecting an action chunk onto these directions yields a vector of coordinates that we call a *command*, which acts as a summary of where the action chunk wants to go. In tandem with the linearity of the stochastic interpolant (Lipman et al., 2023; Albergo et al., 2025), the construction of the command has the property that the commanded movement is invariant along the training path and absent from the regression target, allowing the network to learn only the residual detail given the command rather than the command itself.

We study this factorization strategy on a quadrotor as aerial navigation tasks are well-suited to decomposition, with coarse motion corresponding to net displacement in space and fine motion involving smoothly navigating through obstacles and space. More specifically, we instantiate the command

space as a 16-dimensional space, where each dimension specifies net translation or net yaw along a given axis over one of four temporal windows. Because these commands are physically meaningful, they can be authored by a learned predictor, an external planner, or a person. Furthermore, because factorization yields the ability to denoise atomic motions without explicitly training on them, commands can be treated as executable primitives, so the policy becomes steerable (Wang et al., 2026; Chen et al., 2026) without a need to annotate or acquire any steering vocabulary. We evaluate this through an agent-in-the-loop interface in which an agent can interact with the policy by approving inferred commands or overriding them to perform out-of-distribution tasks.

In summary, our primary contributions are:

- **Coarse and fine factorization of the action chunk.** We split each action chunk into a coarse command and fine motion, and write the command into the source of a flow-matching policy, so the flow learns the fine motion independently of the demonstrated routes.
- **Flexible command authoring.** Linearity of the command space allows the flow to be trained to carry the command unchanged, the command to decode to a readable coarse trajectory before denoising, and the command to be authored by a learned head, a planner, or a person at test time.
- **Real-world evaluation.** On a quadrotor, our policy matches a π_0 baseline on demonstrated tasks on hardware and outperforms it in simulation, flies sketched out-of-distribution routes, and, as a proof of concept, can be piloted by an agent to perform OOD tasks by approving or overriding policy-proposed commands.

2 Related Work

Structured source distributions. Flow and diffusion policies usually map an isotropic Gaussian to actions (Lipman et al., 2023; Chi et al., 2023; Black et al., 2025), yet the source can carry structure: data-dependent priors and couplings speed up training and straighten paths (Lee et al., 2022; Tong et al., 2024), learned or refined noise improves image generation (Ahn et al., 2026; Kim et al., 2026), and phase-preserving diffusion fixes the Fourier phase of the noise so that outputs stay structurally aligned with the input (Zeng et al., 2026). In robot learning, noise has served as a latent action space for steering (Wagenmaker et al., 2025; Tang et al., 2026), and sources have been initialized from action history or proprioception (Jia et al., 2026; Kang et al., 2026; Dai et al., 2026), indexed by latent motion primitives (Machado et al., 2026), or selected for prefix consistency (Ho et al., 2026). These sources are chosen for efficiency or optimized for steering, and the policy still learns the entire action; fine-tuned on few demonstrations, such policies tend to repeat trained behaviours regardless of instruction (Fang et al., 2026; Huang et al., 2026). Following phase-preserving diffusion, we instead fix part of the source such that the coarse component of the chunk is written into the noise, and the flow is trained to carry it and supply the rest of the motion.

Trajectory conditioning and guided sampling. Policies have been conditioned on trajectory sketches (Gu et al., 2023; Sundaresan et al., 2025), point tracks (Wen et al., 2024), VLM-predicted paths (Li et al., 2025), editable visual traces (Lee et al., 2025), and in-context demonstrations (Vosylius & Johns, 2025; Patel et al., 2026), but conditioning is an input the policy may learn to ignore. Other methods leave the policy unchanged and constrain its samples at inference, by denoising from a corrupted guide (Meng et al., 2022), overwriting or projecting intermediate samples (Janner et al., 2022; Wang et al., 2023; Christopher et al., 2024), or steering toward sketches and learned dynamics (Wang et al., 2025; Du & Song, 2025). In contrast, our command is neither an auxiliary conditioning signal nor an inference-time constraint. Rather, because it is embedded directly in the action representation and carried by the flow, a learned head, planner, or human could write to the same command space to be decoded into a coarse trajectory before execution.

3 Background

We consider policies that map an observation o, comprising camera images, proprioceptive state, and language instructions, to a trajectory chunk of H future d-dimensional actions, flattened to $a \in \mathbb{R}^{Hd}$ (Chi et al., 2023; Black et al., 2025). Flow matching (Lipman et al., 2023) trains a continuous-time vector field v_θ to transport a source distribution $p_1(z)$ to the target action distribution $p_0(a)$.

Given a source noise sample $z \sim \mathcal{N}(0, I_{Hd})$ and a ground-truth action chunk $a \sim p_0(a)$, we define the linear stochastic interpolant x_t over time $t \in [0, 1]$ as

$$x_t = tz + (1 - t)a, \qquad t \in [0, 1], \tag{1}$$

which is linear in both endpoints z and a. The velocity along this path is $\frac{\mathrm{d}x_t}{\mathrm{d}t} = z - a$. The flow matching training objective regresses the parameterized velocity field v_θ conditioned on observation o directly onto this vector:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim U[0,1],\, a\sim p_0,\, z\sim\mathcal{N}(0,I)} \left[\|v_\theta(x_t, t, o) - (z - a)\|^2 \right]. \tag{2}$$

At inference, the policy draws an initial source sample $z \sim \mathcal{N}(0, I_{Hd})$ at $t = 1$ and integrates $v_\theta(x_t, t, o)$ backwards from $t = 1$ to $t = 0$ using deterministic Euler numerical integration:

$$x_{t-\Delta t} = x_t - \Delta t \cdot v_\theta(x_t, t, o), \tag{3}$$

yielding an executed action chunk $\hat{a} = x_0$. The robot executes a prefix of $\hat{a}$ and replans asynchronously. We instantiate v_θ building upon π_0 (Black et al., 2025), which uses a vision-language backbone alongside an action expert, fine-tuned on our domain.

4 METHOD

4.1 COMMANDS IN THE SOURCE DISTRIBUTION

Let $U \in \mathbb{R}^{Hd\times K}$ be a matrix with orthonormal columns ($U^\top U = I_K$), defining a K-dimensional linear subspace of the Hd-dimensional action chunk space. We define the coarse command $c \in \mathbb{R}^K$ of an action chunk $a \in \mathbb{R}^{Hd}$ as the coordinates of its projection onto this subspace:

$$c = U^\top a. \tag{4}$$

Applying $U^\top$ to the linear stochastic interpolant $x_t = tz + (1 - t)a$ from Section 3 yields

$$U^\top x_t = tU^\top z + (1 - t)U^\top a. \tag{5}$$

During training, we construct the source noise sample z such that the coordinates of its projection onto the span of U precisely equals the command c of the target data chunk a. Drawing standard Gaussian noise $\epsilon \sim \mathcal{N}(0, I_{Hd})$, we set:

$$z = (I_{Hd} - UU^\top)\epsilon + Uc. \tag{6}$$

Equation 6 replaces the components of ϵ within the command subspace with c, while leaving the orthogonal complement $(I_{Hd} - UU^\top)\epsilon$ as standard Gaussian noise. Substituting Equation 4 and 6 into Equation 5 gives $U^\top x_t = tc + (1 - t)c = c$ for all $t \in [0, 1]$. Consequently, the command is strictly invariant along the entire interpolant path in training.

Furthermore, the flow matching regression target $z - a$ satisfies:

$$U^\top(z - a) = U^\top z - U^\top a = c - c = 0. \tag{7}$$

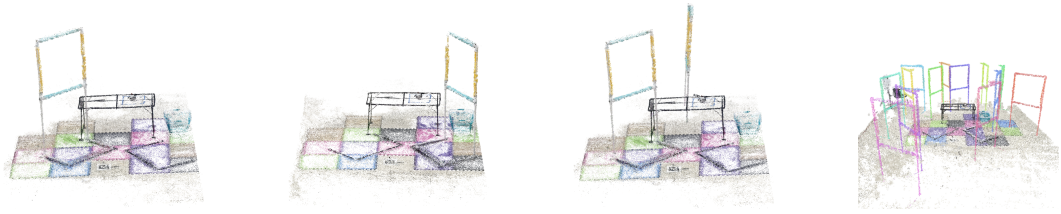
Because the regression target has exactly zero component along the command subspace, the vector field is explicitly trained to maintain $U^\top v_\theta(x_t, t, o) = 0$, making carrying the command a property of the training objective rather than, for instance, an auxiliary loss term or classifier-free guidance heuristic.

At inference, we sample $\epsilon \sim \mathcal{N}(0, I_{Hd})$, substitute any desired command vector $c \in \mathbb{R}^K$ into Equation 6, and integrate $v_\theta(x_t, t, o)$ using deterministic Euler steps (Equation 3).

4.2 CHOICE OF COMMAND SPACE

We instantiate U using prefix-sum displacement functionals over four control channels (forward, lateral, vertical translation, and yaw rate) across four temporal horizons[1]. Gram-Schmidt orthonormalization is applied to the resulting K=16 linear functionals to yield the orthonormal columns of $U \in \mathbb{R}^{Hd\times K}$. Each coordinate of $c = U^\top a$ thus represents the net integrated motion along a specific spatial axis over a defined time window. High-frequency motion details—such as trajectory curvature, velocity profiling, and local obstacle avoidance—occupy the $(Hd - K)$-dimensional orthogonal complement $(I_{Hd} - UU^\top)$ and are dynamically inferred by v_θ from the visual observation o.

[1]Specifically, horizon steps $H_1 = 6, H_2 = 12, H_3 = 25, H_4 = 50$ out of $H = 50$ steps.



(a) Left Gate (b) Right Gate (c) Figure-Eight (d) Arbitrary Gates

Figure 3: **Evaluation scenes as point clouds.** (a, b) The two demonstrated tasks. (c) The figure-eight scene, which adds a center gate that no demonstration visits. (d) The 13 arbitrary gate poses.

4.3 Uncertainty Conditioning

To allow flexible adherence when commands are noisy or conflicting, we train the policy to accept an explicit uncertainty estimate σ. During training, the command component of the source sample z is corrupted by additive noise:

$$z_\sigma = \epsilon - UU^\top \epsilon + U(c + \sigma \cdot \zeta), \qquad \zeta \sim \mathcal{N}(0, I_K), \tag{8}$$

where ζ is scaled per-coordinate by the empirical standard deviation of that command coordinate, and the scalar uncertainty level $\sigma \in [0, \sigma_{\max}]$ is drawn uniformly at random. The scalar σ is provided directly as an additional input to $v_\theta(x_t, t, o, \sigma)$. Crucially, the regression target $z_\sigma - a$ retains its uncorrupted clean action target a, training v_θ to treat c as an observation of the underlying motion with a known noise level σ.

At inference time, setting $\sigma = 0$ recovers the noiseless case of Section 4.1, while setting $\sigma > 0$ provides calibrated slack, allowing the input conditioning to override the command when it contradicts the specified coarse path. When using a learned predictor to author commands, σ is dynamically mapped from the predictor's predictive variance using a monotonic calibration function fitted on held-out demonstration data.

4.4 Test Time Command Authoring

Because the command space consists of K physically meaningful kinematic scalars, commands can be generated by arbitrary upstream authors at test time:

1. **Learned Predictor.** Our policies use a Mixture Density Network (MDN) head trained jointly on top of the vision-language backbone representations. At inference, commands are selected from the dominant mixture component.
2. **Interactive Route Sketching.** Human operators can author commands directly via a 2D sketch interface. A polyline sketched on a camera view is resampled by arc length at the robot's nominal speed, converted to a synthetic action track, and projected into command space via $c_{\text{sketch}} = U^\top a_{\text{synth}}$ over a sliding window anchored at the vehicle's nearest path point. Closed-loop lateral tracking is maintained using a decay-curve rejoin construction.
3. **Agent in the Loop.** At each replan, an agent sees the camera images and the Learned Predictor's command decoded as a coarse trajectory Uc, and either approves it or replaces it with a relative displacement, which is projected into command space in the same way as a sketch.

5 Experiments

We design the experiments to answer four questions. (1) Does splitting the chunk into a command and a residual cost anything on the demonstrated tasks, and does it help under visual shift (Section 5.2)? (2) How well does uncertainty work (Section 5.3)? (3) Can a person steer the policy along routes that demonstrations do not cover (Section 5.4)? (4) Can an agent pilot the policy through the command channel (Section 5.5)?

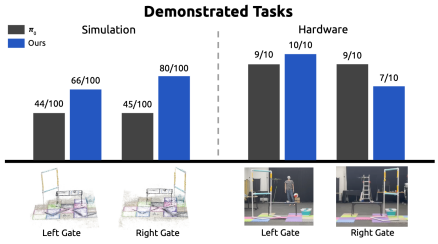


Figure 4: **Demonstrated tasks.** Success of π_0 and ours, both trained on the same 100 real demonstrations. Simulated images differ from the training images, so simulation also measures robustness to visual shift.

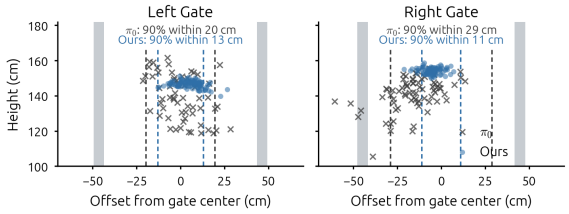


Figure 5: **Gate crossings in simulation.** Each marker is one crossing of the gate plane. Dashed lines enclose 90% of each policy's crossings, and gray bars mark the gate posts.

5.1 Experimental Setup

Platform and simulation. The policy controls a quadrotor from onboard camera images. An action chunk is $H = 50$ steps of per-step displacements $(\Delta x, \Delta y, \Delta z)$ and yaw increments $\Delta\psi$ in the motion-capture frame (Tucker et al., 2026). Simulation renders a 3D Gaussian Splatting (GS) reconstruction (Kerbl et al., 2023) of the flight room in the four layouts of Figure 3. The two single-gate layouts match the room in which the demonstrations were collected, and the other two are built with GS editing (Shorinwa et al., 2024).

Policies. The training set contains 100 real demonstrations, 50 through the left gate and 50 through the right gate, flown by a human pilot. We fine-tune two policies from the same π_0 checkpoint for 5,000 steps: a standard fine-tune, labeled π_0, and a fine-tune with the command channel, labeled *ours*. Unless stated otherwise, ours sets σ from the head's predictive variance (Section 4.3). All sketch and agent experiments use our checkpoint.

Baselines. On the demonstrated tasks we compare against the π_0 fine-tune. For sketched routes, we compare against three ways of imposing the same command c_{sketch} on the π_0 fine-tune. *Injected sketch* writes c_{sketch} into the source noise with Equation 6 and changes nothing else. *SDEdit* (Meng et al., 2022) partially noises the sketch's action track to $x_{t_0} = t_0\,\epsilon + (1 - t_0)\,a_{\text{sketch}}$ with $t_0 = 0.5$ and denoises from there. *Velocity projection* injects the command in the same way and also removes the command component of the predicted velocity at every integration step, $v_\theta \leftarrow (I_{Hd} - UU^\top)\,v_\theta$, so that $U^\top x_t = c_{\text{sketch}}$ holds exactly throughout denoising. In the agent experiments, the baseline executes the agent's primitives directly as waypoints, without the policy.

Metrics. We model the drone as a sphere with radius 18 cm. A flight succeeds when the sphere crosses the plane of each gate inside its opening, in the correct direction and order, without touching a post. A failure either misses the opening or touches a post. A sketched route is completed when it is flown to its end, and its tracking error is the mean distance between the flown path and the sketch, taken as a median over flights.

Protocol. In simulation we run 100 flights per demonstrated gate, 5 flights at each of the 13 arbitrary gate poses, and 10 flights each for the orbit and the figure-eight. On hardware we run 10 flights per demonstrated gate and 5 flights each for the orbit and the figure-eight. Methods compared on the same sketched route use the same sketch and seeds.

5.2 Task Success on Demonstrated Tasks

We first ask whether splitting the chunk into a command and a residual hurts the tasks the policy was trained on by evaluating in two settings. On hardware, the observations match the training distribution. In simulation, the Gaussian-splat renderings introduce a visual shift. The splat is reconstructed from a handheld phone capture rather than from the drone's own cameras, so the rendered views differ from the training images in color, sharpness, and exposure, and contain reconstruction artifacts such as floaters and blurred texture where the capture was sparse (Figure 8).

On hardware, both policies are near ceiling, with 18/20 flights for π_0 and 17/20 for ours, a difference within the variance of 10 trials per gate (Figure 4). In simulation the gap opens, where ours succeeds in 146/200 flights and π_0 in 89/200. Most of the gain comes from fewer missed openings, 19 for ours against 69 for π_0. Our crossings are also closer to the center of the opening (Figure 5): 90% of them fall within 13 cm of the center at the left gate and 11 cm at the right gate, against 20 cm and 29 cm for π_0.

We attribute the gain to two properties of the command channel. First, the head predicts a 16-dimensional command instead of a full action chunk. A low-dimensional summary of coarse motion is easier to predict from shifted observations, and since errors in coarse motion are what carry the drone into states the demonstrations never visited, fewer of them limit compounding covariate shift. This agrees with the drop in missed openings. Second, the trust input σ lets the flow correct commands the head is unsure of instead of following them into a failure, which we examine next.

5.3 Effect of the Trust Input σ

We ablate the uncertainty input σ to measure its effect on task success. To isolate the role of σ, we rerun the demonstrated tasks in simulation with σ fixed instead of set by the head, using the same 100 flights per gate and the same seeds as Figure 4. Both fixed settings fail, in opposite ways (Figure 6).

With $\sigma = 0$, the policy treats every command as reliable and tracks it as closely as it can. When the head's command is wrong under visual shift, the policy follows it and touches the left gate's posts in 69 of 100 flights. Total success drops to 87/200, no better than π_0, so predicting the command is not enough on its own. With σ at its cap of 1.5, the command barely steers the flight and the policy misses the gates in most flights, reaching only 51/200. The flow was trained to rely on the command for the route and does not recover the route from the image alone.

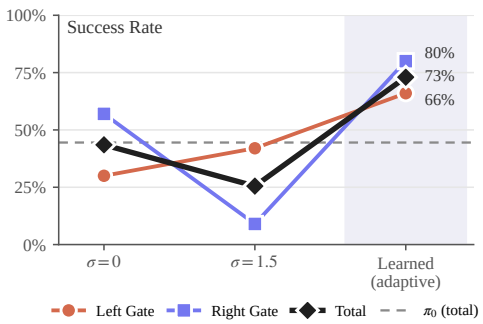


Figure 6: **Trust input.** Simulated success on the demonstrated tasks with σ fixed or set by the head. Dashed line: total success of π_0.

Setting σ from the head's predictive uncertainty lies between these extremes: the policy follows confident commands and relies on the image where the head is uncertain. The gain in Section 5.2 therefore needs both parts, predicting the coarse motion and letting the flow correct it when the prediction is unreliable.

5.4 Out-of-Distribution Route Following

We next ask whether the policy can fly routes that no demonstration covers. A person draws each route with the sketch interface (Section 4.4), and the resulting command c_{sketch} is given to the policy without further tuning.

Injecting the command into the source of π_0 is not enough. It completes 24/65, 2/10, and 1/10 routes (Figure 7) and drifts tens of centimeters from the sketch (Table 1), because a flow that was never trained to carry a command in its source largely ignores it. Ours was trained to carry the command and supply only the residual motion. It completes 60/65 arbitrary-gate flights, where all five failures are slight grazes of a post, and every orbit and figure-eight flight. On hardware it completes all 5 orbit and all 5 figure-eight flights (Figure 7).

Methods that enforce the command more strongly at inference also succeed. Velocity projection completes every route with under 2 cm of tracking error, and SDEdit completes most of them (Table 1). All four methods use the same command space and differ only in how they impose the command during denoising, so the command space by itself is enough to steer the flow along a good sketch. Ours tracks the sketch less tightly because σ leaves it room to deviate from the command, which is relevant in two cases. Without an author, ours predicts its own command (Section 5.2). Given a flawed sketch that passes 7 cm from a gate post, a version of ours trained with more data

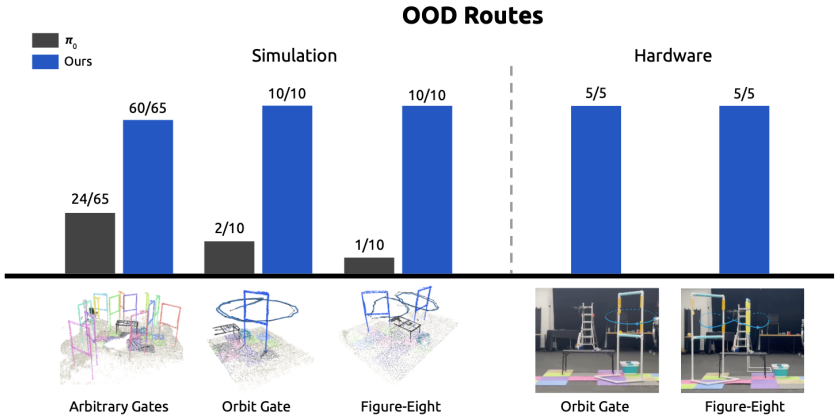
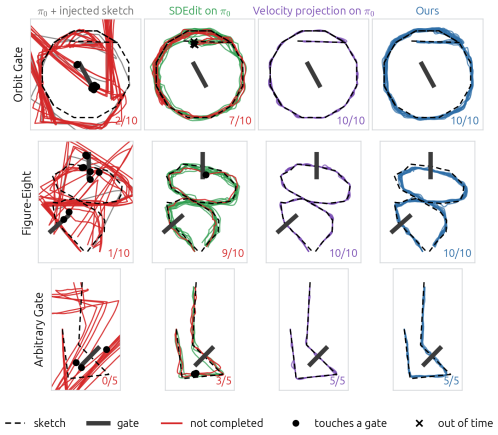


Figure 7: **OOD routes.** Success on sketched routes that no demonstration covers. For π_0, the sketched command is written into its source noise (Equation 6). Arbitrary Gates flies 5 trials at each of the 13 gate poses in Figure 3, and Orbit Gate flies in the Right Gate scene. Hardware flights were run for ours only.

Table 1: **Ways of imposing a sketched command, in simulation.** Completed routes and tracking error (median over flights). All baselines impose the command on the π_0 fine-tune of Figure 4. Left: flown paths per method; red flights did not complete the route, dots mark gate contact, and crosses mark flights that ran out of time.



	Arbitrary Gates	Orbit Gate	Figure-Eight
Completed			
π_0 + sketch	24/65	2/10	1/10
SDEdit	58/65	7/10	9/10
Vel. Proj.	**65/65**	**10/10**	**10/10**
Ours	60/65	**10/10**	**10/10**
Tracking error (cm)			
π_0 + sketch	26.8	76.2	43.7
SDEdit	5.4	4.9	6.1
Vel. Proj.	**0.9**	**0.9**	**1.3**
Ours	4.9	4.4	6.1

moves away from the sketch and clears the post, while velocity projection and SDEdit follow the sketch into it (Appendix D).

5.5 Agent as Command Author

Finally, we ask whether an agent with no experience of the platform can pilot the policy through the command channel. The agent receives a written task description, a small set of movement primitives, and a budget of moves. At each replan, it sees the camera images and the policy's own proposal decoded as a coarse trajectory $U\hat{c}$, where $\hat{c}$ is the command predicted by the head. It either approves the proposal or emits a primitive, which is converted to a command in the same way as a sketch (Section 4.4). Both prompts are given in Appendix C. We evaluate Claude Sonnet 5 and Claude Opus 5 in simulation on three tasks that no demonstration covers.

Through the command channel, both agents complete every task at least twice (Table 2). With the same agent, the command channel succeeds in 7/15 trials and waypoints in 5/15, and Opus raises success to 11/15. The command channel helps most on the tasks that require finding the mannequin

Table 2: **Agent as command author.** Success in simulation over 5 trials per task. The agent pilots the drone either through our command channel, approving or overriding the policy's proposed command, or by issuing waypoints directly to the controller.

	Claude Sonnet 5		Claude Opus 5
Task	Waypoints	Ours	Ours
Find the mannequin (start facing away)	0/5	3/5	4/5
One circle around the center gate	4/5	2/5	3/5
Left gate, then find the mannequin	1/5	2/5	4/5
Total	5/15	7/15	**11/15**

(3/5 vs. 0/5, and 2/5 vs. 1/5). There the agent only has to decide where to go and can leave the approach to the policy's trained behavior. Waypoints do better on circling the center gate (4/5 vs. 2/5), a purely geometric route where specifying each move directly is enough. On hardware, Sonnet completes *left gate, then find the mannequin* in both of two trials.

Most failures come from the agent's own decisions rather than from how the policy executes its commands, for example misjudging which side of the frame the mannequin is on or how far away it is (Figure 9). This agrees with the gain from Sonnet to Opus: success rises with the agent's capability while the command channel stays the same.

6 Discussion and Conclusion

We proposed writing the coarse component of an action chunk into the source noise of a flow-matching policy, so that carrying the command becomes a property of the training objective and the flow learns only the fine motion. Trained on 100 real demonstrations of two gates, the policy raises simulated success under visual shift from 89/200 to 146/200 over a π_0 fine-tune, and flies unseen orbits, figure-eights, and gates from a sketched command in simulation and on hardware. These results show that with few demonstrations, separating where to go from how to fly lets the policy reuse the motion it learned on new routes, and that predicting the route as a low-dimensional command is more robust than predicting it inside a full action chunk. The command also gives the policy a practical interface: it can be written by a learned head, a person, or an agent, decoded into a coarse trajectory for inspection before execution, and followed loosely through σ when it is unreliable.

Limitations and future work. We instantiate the command space for quadrotor flight, where net translation and yaw over a few time windows is a natural summary of coarse motion, and evaluate on gate navigation. The construction itself only requires a linear map U from action chunks to a low-dimensional subspace, so other embodiments such as manipulators can use the same training objective with a different U, for example one derived from the principal components of demonstration actions. We leave a systematic study of this choice to future work. Our agent-in-the-loop interface is also synchronous: the drone holds its position while the agent decides on each command, which takes a median of around 15 seconds per decision. Because an approved command is simply the policy's own proposal, this latency could be hidden by running the agent asynchronously while the policy continues to fly. More broadly, we view the source distribution as a natural interface between generative robot policies and the people, planners, and agents that direct them.

7 AI DISCLOSURE

In this work, we used generative AI tools (Claude, Anthropic, through Claude Code) to implement methods and experiments designed by the authors, to help refine hypotheses, and to assist in interpreting results, with all final interpretations made by the authors. We have not used generative AI tools to formulate mathematical claims, develop the conceptual framework, or design the methodology and experiments, and proofs, synthetic data generation, translation, dataset cleaning, and qualitative analysis are not applicable to this work. Additionally, generative AI tools facilitated figure creation and the writing of the paper. We have reviewed all AI-assisted work. All reported numbers were produced by scoring scripts that the authors inspected, with trajectories verified visually in point-cloud views, and hardware outcomes were judged by human observers. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Donghoon Ahn, Jiwon Kang, Sanghyun Lee, Jaewon Min, Minjae Kim, Wooseok Jang, Hyoungwon Cho, Sayak Paul, SeonHwa Kim, Eunju Cha, Kyong Hwan Jin, and Seungryong Kim. A noise is worth diffusion guidance. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=xEWooSOgaz.

Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions, 2025. URL https://arxiv.org/abs/2303.08797.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

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

William Chen, Jagdeep Singh Bhatia, Catherine Glossop, Nikhil Mathihalli, Ria Doshi, Andy Tang, Danny Driess, Karl Pertsch, and Sergey Levine. Steerable vision-language-action policies for embodied reasoning and hierarchical control. *CoRR*, abs/2602.13193, 2026. doi: 10.48550/ARXIV.2602.13193. URL https://doi.org/10.48550/arXiv.2602.13193.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.026.

Jacob K Christopher, Stephen Baek, and Ferdinando Fioretto. Constrained synthesis with projected diffusion models. In *Neural Information Processing Systems*, 2024.

Meipo Dai, Qiyuan Zhuang, He-Yang Xu, Ying-Jie Shuai, Yijun Wang, Qi Dou, and Xiu-Shen Wei. Where should action generation begin? a learnable source prior for generative robot policies, 2026. URL https://arxiv.org/abs/2606.17408.

Maximilian Du and Shuran Song. DynaGuide: Steering diffusion policies with active dynamic guidance. In *Proceedings of the 39th Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Yu Fang, Yuchun Feng, Dong Jing, Jiaqi Liu, Yue Yang, Zhenyu Wei, Daniel Szafir, and Mingyu Ding. When vision overrides language: Evaluating and mitigating counterfactual failures in vlas, 2026. URL https://arxiv.org/abs/2602.17659.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In Dana Kulic, Gentiane Venture, Kostas E. Bekris, and Enrique Coronado (eds.), *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://doi.org/10.15607/RSS.2024.XX.090.

Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches, 2023.

Chengyang He, Xu Liu, Gadiel Mark Sznaier Camps, Joseph Bruno, Guillaume Adrien Sartoretti, and Mac Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *The Fourteenth International Conference on Learning Representations*, 2026.

Trong-Bao Ho, Quang-Tan Nguyen, Thien-Loc Ha, Gia-Binh Nguyen, Viet-Thanh Nguyen, Long Dinh, Minh N. Vu, Duy M. H. Nguyen, An Thai Le, and Ngo Anh Vien. Start right, arrive right: Asynchronous execution via initial noise selection, 2026. URL https://arxiv.org/abs/2606.19774.

Suning Huang, Jiaqi Shao, Ke Wang, Qianzhong Chen, Jiankai Sun, Yanjiang Guo, Mac Schwager, and Jeannette Bohg. Breaking lock-in: Preserving steerability under low-data vla post-training, 2026. URL https://arxiv.org/abs/2604.23121.

Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

Jindou Jia, Gen Li, Xiangyu Chen, Tuo An, Yuxuan Hu, Jingliang Li, Xinying Guo, and Jianfei Yang. A2A: Action-to-Action Flow Matching Policy. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026.

Sinjae Kang, Chanyoung Kim, Kaixin Wang, Li Zhao, and Kimin Lee. Warmprior: Straightening flow-matching policies with temporal priors, 2026. URL https://arxiv.org/abs/2605.13959.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139:1–139:14, 2023. doi: 10.1145/3592433. URL https://doi.org/10.1145/3592433.

Junwan Kim, Jiho Park, Seonghu Jeon, and Seungryong Kim. Better source, better flow: Learning condition-dependent source distribution for flow matching, 2026. URL https://arxiv.org/abs/2602.05951.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Paul Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli VanderBilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. Molmoact: Action reasoning models that can reason in space. *CoRR*, abs/2508.07917, 2025. doi: 10.48550/ARXIV.2508.07917. URL https://doi.org/10.48550/arXiv.2508.07917.

Sang-gil Lee, Heeseung Kim, Chaehun Shin, Xu Tan, Chang Liu, Qi Meng, Tao Qin, Wei Chen, Sungroh Yoon, and Tie-Yan Liu. PriorGrad: Improving conditional denoising diffusion models with data-dependent adaptive prior. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=_BNiN4IjC5.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical action models for open-world robot manipulation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=h7aQxzKbq6.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can RL bring to VLA generalization? an empirical study. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=qmBMPInbZC.

Bruno Machado, Alexandre Chapin, Emmanuel Dellandréa, and Liming Chen. Flowing with purpose: Latent action guided flow matching policies for robotic manipulation. *ArXiv*, abs/2606.23420, 2026. URL https://api.semanticscholar.org/CorpusID:289622254.

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 *International Conference on Learning Representations*, 2022.

Austin Patel, Ben Pekarek, Joel Enrique Castro Hernandez, and Shuran Song. Behavior prompting policy: Demonstrations as prompts for manipulation. *arXiv preprint arXiv:2606.30457*, 2026. URL https://arxiv.org/abs/2606.30457.

Ola Shorinwa, Johnathan Tucker, Aliyah Smith, Aiden Swann, Timothy Chen, Roya Firoozi, Monroe Kennedy III, and Mac Schwager. Splat-mover: Multi-stage, open-vocabulary robotic manipulation via editable gaussian splatting. *arXiv preprint arXiv:2405.04378*, 2024.

Priya Sundaresan, Quan Vuong, Jiayuan Gu, Peng Xu, Ted Xiao, Sean Kirmani, Tianhe Yu, Michael Stark, Ajinkya Jain, Karol Hausman, Dorsa Sadigh, Jeannette Bohg, and Stefan Schaal. Rt-sketch: Goal-conditioned imitation learning from hand-drawn sketches. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 70–96. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/sundaresan25a.html.

Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, et al. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–12, 2023.

Andy Tang, William Chen, Andrew Wagenmaker, Chelsea Finn, and Sergey Levine. Improving robotic generalist policies via flow reversal steering. 2026.

Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=CD9Snc73AW. Expert Certification.

Johnathan Tucker, Denis Liu, Aiden Swann, Allen Ren, Javier Yu, Jiankai Sun, Brandon Kim, Lachlain McGranahan, Quan Vuong, and Mac Schwager. π, but make it fly: Physics-guided transfer of vla models to aerial manipulation, 2026. URL https://arxiv.org/abs/2603.25038.

Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados—a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14(1):147–183, 2022.

Vitalis Vosylius and Edward Johns. Instant policy: In-context imitation learning via graph diffusion. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. *Conference on Robot Learning*, 2025.

Maggie Wang, Lars Osterberg, Stephen Tian, Ola Shorinwa, Joseph Bowkett, Daniel Pastor, Jiajun Wu, and Mac Schwager. Insight: Self-guided skill acquisition via steerable vlas, 2026. URL https://arxiv.org/abs/2606.24884.

Yanwei Wang, Lirui Wang, Yilun Du, Balakumar Sundaralingam, Xuning Yang, Yu-Wei Chao, Claudia Pérez-D'Arpino, Dieter Fox, and Julie A. Shah. Inference-time policy steering with human interactions. In *International Conference on Robotics and Automation (ICRA)*, May 2025.

Yinhuai Wang, Jiwen Yu, and Jian Zhang. Zero-shot image restoration using denoising diffusion null-space model. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=mRieQgMtNTQ.

Chuan Wen, Xingyu Lin, John Ian Reyes So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point trajectory modeling for policy learning. In Dana Kulic, Gentiane Venture, Kostas E. Bekris, and Enrique Coronado (eds.), *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi: 10.15607/RSS.2024.XX.092. URL https://doi.org/10.15607/RSS.2024.XX.092.

Congrong Xu, Justin Kerr, and Angjoo Kanazawa. Splatfacto-w: A nerfstudio implementation of gaussian splatting for unconstrained photo collections. *arXiv preprint arXiv:2407.12306*, 2024.

Yu Zeng, Charles Ochoa, Mingyuan Zhou, Vishal M. Patel, Vitor Guizilini, and Rowan McAllister. NeuralRemaster: Phase-preserving diffusion for structure-aligned generation, 2026. URL https://arxiv.org/abs/2512.05106.

Jiaming Zhou, Ke Ye, Jiayi LIU, Teli Ma, Zifan Wang, Ronghe Qiu, Kun-Yu Lin, Zhilin Zhao, and Junwei Liang. Exploring the limits of vision-language-action manipulation in cross-task generalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=h6xQClTm4W.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization, 2026. URL https://arxiv.org/abs/2510.03827.

A HARDWARE

All real-world experiments use a ModalAI Starling 2 Max quadrotor flown in a motion-capture flight room. The policy observes two onboard cameras, a forward-facing and a downward-facing fisheye camera, together with the drone's pose (x, y, z, ψ) from the motion-capture system.

The policy runs off-board on a workstation with an NVIDIA RTX 4090 GPU. Camera images are streamed from the drone to the workstation over Wi-Fi via ROS 2 at 1024×768, consumed at 5 Hz. Each inference produces an action chunk of $H = 50$ per-step displacements $(\Delta x, \Delta y, \Delta z, \Delta\psi)$ expressed in the motion-capture frame. The chunk is sent back over ROS 2, integrated into a sequence of position and yaw setpoints, and tracked by the PX4 controller running on the onboard VOXL 2 at 100 Hz.

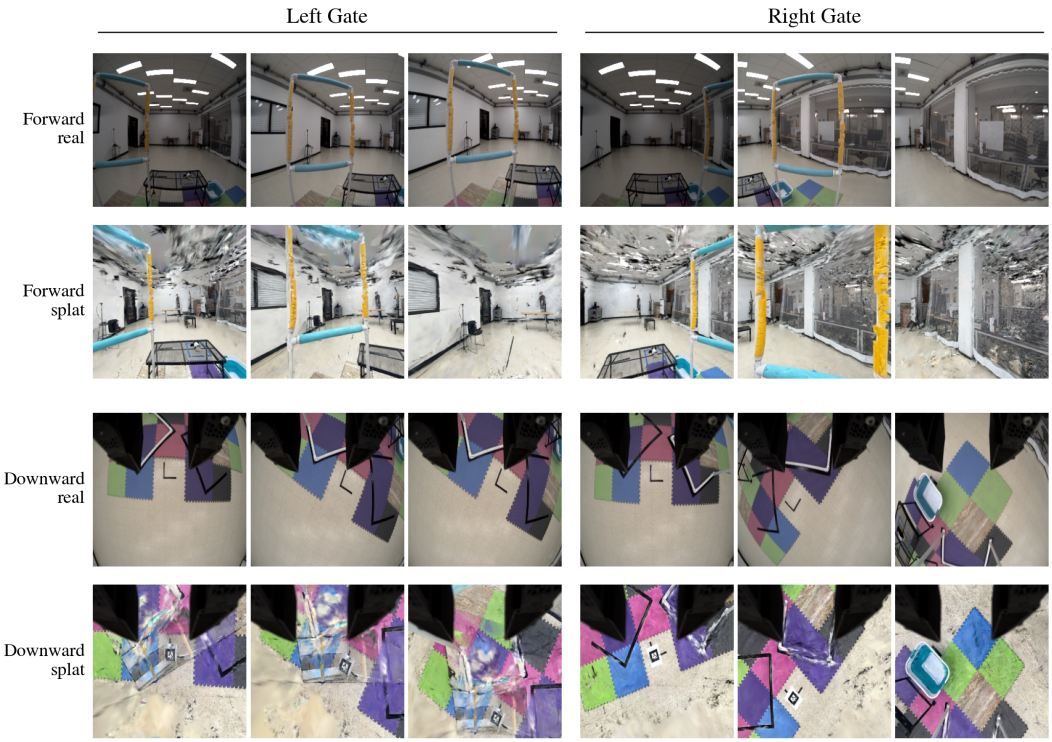


Figure 8: **Real vs. simulated observations.** Forward and downward camera images from real demonstrations of the left gate (first three columns) and right gate (last three columns), next to Gaussian splat renders from the same pose.

B Gaussian Splat Environments

Scene capture and reconstruction. We reconstruct the flight room as a 3D Gaussian Splatting (3DGS) model (Kerbl et al., 2023), following the pipeline of (Tucker et al., 2026). Unlike that work, which captures the scene with the drone's own forward camera, we record a handheld walk-through of the room with an iPhone using the Record3D app, which logs each RGB frame together with the ARKit camera pose and intrinsics. Because ARKit fuses visual and inertial measurements, these poses are metrically scaled. We convert the capture with Nerfstudio's Record3D data processor and train the splat with splatfacto (Tancik et al., 2023; Xu et al., 2024) for the default number of iterations, so no structure-from-motion step is needed.

Figure 8 compares real training images with Gaussian splat renders from the same poses. Renders use a pinhole camera model, while the real cameras are fisheye, and the reconstructions contain artifacts such as floaters on the ceiling, ghosting on the floor, and a different floor layout than during the demonstrations.

Alignment to the motion-capture frame. The Record3D poses are metric but expressed in an arbitrary ARKit world frame, so the reconstruction differs from the motion-capture frame only by a rigid transform. To recover it, we place an ArUco marker at a known pose T_A^M in the motion-capture frame at the motion-capture origin, axis-aligned with it. In capture frames where the marker is visible, we estimate its pose relative to the camera from its four detected corners via PnP with the recorded intrinsics and compose this with the ARKit camera pose to obtain the marker pose T_A^C in the capture frame. The capture-to-mocap transform is then $T_C^M = T_A^M\,(T_A^C)^{-1}$. We apply T_C^M to all camera poses before training, so the resulting splat lies directly in the metric motion-capture frame. Poses, actions, commands and sketches are therefore interchangeable between simulation and hardware without further calibration.

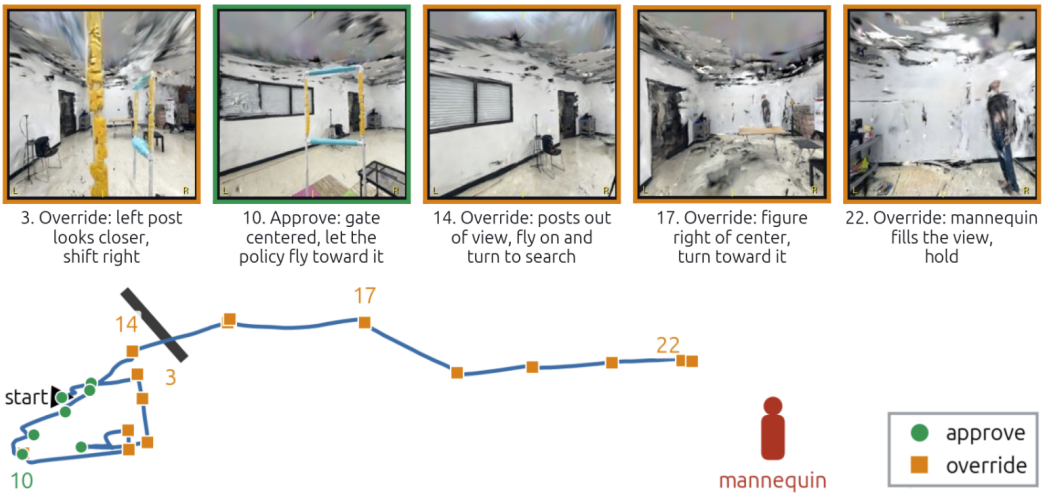


Figure 9: **Agent in the loop.** A simulated flight in which Sonnet flies through the left gate and finds the mannequin, seen from above. Markers show each decision (approve or override); insets show the forward view at key decisions.

Closed-loop simulation. Action chunks are executed with the same interface as on hardware (Appendix A): the per-step displacements are integrated into position and yaw setpoints and tracked by a controller acting on a semi-kinematic quadrotor model with state (p^W, v^W, q_R^W), simulated in ACADOS (Verschueren et al., 2022). For the mixed-data experiments in Appendix D, we additionally generate simulated demonstrations by planning collision-free trajectories through the gates in these environments, with randomization of the initial state and waypoints.

Scene editing and success checking. For the edited layouts of Figure 3, we segment the Gaussians belonging to a physical gate, then duplicate and rigidly transform them to each target pose. As each gate's pose is known exactly in the motion-capture frame, the same gate geometry is used to score flights: the drone is modeled as an 18 cm sphere, and a gate counts as passed when this sphere crosses the gate's plane inside its opening, in the correct direction and order, without intersecting a post.

C Agentic Experiment Prompts

You are an agent controlling a quadrotor drone in a motion-capture flight room. Your goal is to fly the drone through the gate, locate the mannequin in the scene, and then fly up to the mannequin. Please be mindful of safety — we do not want any collisions with the environment. You can use the motion primitives below to move the drone as you see fit, leveraging the observations. A safety pilot is watching and can take over at any time.

Every decision is flown by a fine-tuned policy that was trained in this room to fly through the gate and hover over the stuffed penguin. You are always steering that policy, never the motors: a motion primitive is the command the policy is given for that decision, and `--auto` hands the decision to the policy's own trained behaviour instead. Because the policy realises your command rather than replaying it, expect roughly what you asked for and not exactly — usually a little less. The readout gives your pose before and after every decision; use it to see what you actually got.

THE LOOP. Run these from `~/code/gate-pin` with `python3`. The drone holds position while you think; nothing moves until you act.

```
python3 tools/agent_cli.py wait --dir {DIR}
python3 tools/agent_cli.py act  --dir {DIR} --k K --say "one sentence" [primitives | --auto]
--then-wait
python3 tools/agent_cli.py stop --dir {DIR} --k K --say "one sentence"
```

Call plain `wait` once, for the first decision; after that `--then-wait` submits your decision and prints the next one in the same call. `--k` must match the decision you were shown. `--say` is required and is recorded with the flight. `wait` prints "flight ended" when the flight is over.

WHAT YOU SEE. Each decision prints the path of ONE image; read it with the Read tool. Its LEFT half is the forward camera, its RIGHT half the downward camera. The forward panel has a yellow "L" in its bottom-left corner, a yellow "R" in its bottom-right, and a yellow tick at the top and bottom of its centre column; say which mark something is nearer when you describe it. Both cameras are raw fisheye, so straight edges bow. The readout also gives your pose, your previous decisions, and the poses you have been at.

For reference, read these two images once before you start:

```
~/code/gate-pin/docs/gate_reference.png        the gates, in this room, from this camera
~/code/gate-pin/docs/mannequin_reference.png   the mannequin
```

PRIMITIVES. One move per decision, executed over about 5 seconds; combine them freely.

```
--forward M   metres along your heading          --yaw DEG   degrees, positive turns left
--left M      metres to your left                --scan DEG  a pure turn, no translation
--up M        metres up                          --dx/--dy/--dz M  the same move in fixed room
 axes
--auto        no move: the policy chooses this decision itself from its trained behaviour. Put
 the
              instruction it was trained on in --say, e.g. "go through the gate on the left
and hover
              over the stuffed animal" (the same sentence for the right gate also works).
```

Limits per decision: 2 m horizontally, 1 m vertically, 45 degrees of yaw, altitude held to 0.79–1.75 m. A move beyond a limit is clamped and you are told.

THE FRAME. Pose is $[x, y, z, \text{heading}]$, metres and radians, in fixed room axes that never rotate. Heading 0 faces $+x$, $+1.57$ faces $+y$, $+3.14$ faces $-x$. Positive yaw increases the heading.

When you are done, or if the pilot stops you, `stop` and write a short report: what you saw, what you did, where you ended up, and whether you believe you completed the task.

D TRAINING WITH MORE DATA

Table 3: **Training with more data.** Both policies are fine-tuned on real and simulated demonstrations. Demonstrated tasks pool two training seeds; on OOD sketches π_0 receives the same command injected into its source.

	Demonstrated				OOD Sketches		
Policy	Left Gate	Right Gate	Center (from Left)	Center (from Right)	Left then Center	Orbit Gate	Figure-Eight
π_0 (plain fine-tune)	17/20	19/20	16/20	7/20	0/5	0/5	0/5
ours (channel + head)	**20/20**	**20/20**	**20/20**	**16/20**	**5/5**	**5/5**	**5/5**

Our main policy is trained on 100 real demonstrations of two straight gate passes. To study what the factorization gains with more data, we also train both policies on a mixed corpus of real and simulated demonstrations covering more routes, including a new set of center gate tasks. On the demonstrated gates, our policy succeeds on 76 of 80 flights across two training seeds against 59 for π_0. Given sketched routes, it completes a hand-drawn left-then-center route, an orbit, and a figure-eight in every trial, while π_0 with the same command injected into its source completes none. With more data, the trust input σ also lets the policy correct a flawed sketch (Figure 10): on a left-then-center sketch that passes 7 cm from the center gate's post, our policy at $\sigma = 0.5$ completes 8 of 10 flights and clears the post by 21 cm, while velocity projection follows the sketch and touches the post in all 10.

Velocity projection and SDEdit both fail on this sketch. Velocity projection follows it into the post, even when part of the flow's own velocity is kept, and SDEdit either touches the post or leaves the route (Table 4). Our policy, given room to deviate through σ, instead flies a smooth path around the post and completes the route in 8 of 10 flights.

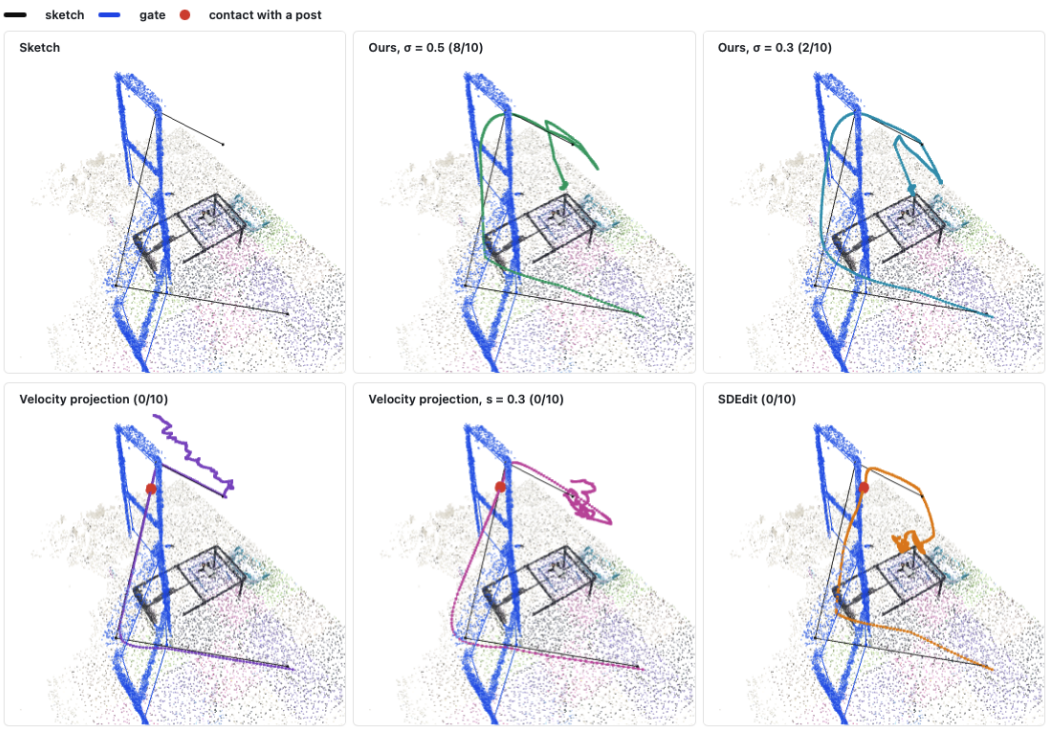


Figure 10: **Correcting a dangerous sketch.** A left-then-center sketch that passes 7 cm from the center gate's post, flown by the policies trained with more data. Each panel shows one representative flight and, in parentheses, how many of 10 flights completed the route. Red marks where a flight touches a post. Our policy at $\sigma = 0.5$ moves away from the flawed sketch and clears the post, while velocity projection follows the sketch into the post even when part of the flow's own velocity is kept ($s = 0.3$), and SDEdit touches the post and drifts from the route.

Table 4: **Flawed sketch.** A left-then-center sketch that passes 7 cm from the center gate's post, flown by the policies trained with more data. For velocity projection, s is the fraction of the flow's own velocity along the command that is kept ($s = 0$ imposes the command exactly). Closest approach is the median over 10 flights.

Method	Trust	Completed	Closest to the Post (cm)
ours	$\sigma = 0$	0/10	3.4
	$\sigma = 0.3$	2/10	8.2
	$\sigma = 0.5$	**8/10**	**21.1**
Velocity projection on π_0	$s = 0$	0/10	6.3
	$s = 0.1$	0/10	11.7
	$s = 0.3$	0/10	6.0
	$s = 0.5$	0/10	7.5
SDEdit on π_0	–	0/10	3.2