SnapFlow: One-Step Action Generation for Flow-Matching VLAs via Progressive Self-Distillation

Anonymous authors
Paper under double-blind review

Abstract

Vision-language-action (VLA) models with flow-matching action heads, such as $\pi 0$ and $\pi 0.5$, have become a standard recipe for generalist robot manipulation, but every action chunk is produced by integrating a learned velocity field over about ten sequential action-expert evaluations, so action generation dominates model latency and caps the achievable control frequency. Existing remedies either shrink each evaluation through layer distillation or pruning while leaving the sequential solver in place, or import one-step objectives from image generation that were designed for training from scratch, rely on an exponential-moving-average teacher or an external critic, and regress finite-span targets onto the conditional velocity, whose variance around the marginal field biases the learned map. We introduce SnapFlow, a self-distillation method that adapts a pretrained flow-matching VLA to one-step action generation. SnapFlow mixes local flow-matching supervision with shortcut samples whose target is the model's own two-step Euler endpoint under stop-gradient, and adds a single zero-initialized destination-time projection so that one action expert serves both objectives while the pretrained predictor is preserved at initialization; no separate teacher network or critic is required. On $\pi 0.5$, whose 10-NFE teacher operates near ceiling, the archived one-NFE SnapFlow checkpoint reaches 98.75% LIBERO success versus 97.75% for the teacher; under a separate paired evaluation of $\pi 0$, the one-NFE model surpasses its teacher by 10.5 percentage points. SnapFlow also lowers mean and tail action error on held-out data, cuts batch-one model latency by about 3.3×, and transfers to shelf retrieval on a dual-arm humanoid.

1 Introduction

Vision-Language-Action (VLA) models (Black et al., 2024; Physical Intelligence et al., 2025) have advanced generalist robotic manipulation by pairing a vision-language backbone with an action expert that maps observations and instructions to action chunks. The strongest open recipes, $\pi 0$ (Black et al., 2024) and $\pi 0.5$ (Physical Intelligence et al., 2025), generate these chunks via *flow matching* (Lipman et al., 2023): a learned velocity field iteratively denoises Gaussian noise into a coherent action chunk through 10 Euler steps, i.e. 10 neural function evaluations (NFE) of the action expert, while the VLM prefix is computed once per chunk. Because these evaluations are sequential, the action expert rather than the backbone dominates model latency, and a single inference already occupies most of the control period of a low-rate controller before communication and execution overhead are counted (Appendix G quantifies this on a datacenter GPU). This latency is not a mere inconvenience: it bounds how often the policy can observe and correct, so contact-rich grasps, moving targets, and recovery from disturbances all depend on shortening the loop, and on modest onboard hardware the same ten evaluations take proportionally longer.

The simplest way to remove this cost is to run the pretrained velocity field with a single Euler step, but a field trained for local updates is then asked to produce the full noise-to-action displacement. Whether this hurts depends on the field itself: a well-converged field is

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.

nearly straight and tolerates the regime change, whereas a curved, under-converged field accumulates integration error at every step, so removing steps can even help. Existing acceleration methods do not resolve this tension. Architecture- and system-level approaches, including layer distillation (Jeon et al., 2026), layer and token pruning (Yang et al., 2025), and optimized runtimes (Nguyen et al., 2026), reduce the cost of each action-expert evaluation but keep all ten of them. Fast-sampling objectives from generative modeling, such as consistency models (Song et al., 2023), shortcut models (Frans et al., 2025), and MeanFlow (Geng et al., 2025), and their robot-policy adaptations (Prasad et al., 2024; Wang et al., 2025; Zhang et al., 2025; Yan et al., 2025), were designed for training from scratch and typically depend on an exponential-moving-average teacher, a Jacobian-vector product through the network, or an external critic at deployment (Yang et al., 2026), all costly for a billion-parameter VLA. Moreover, they build finite-span consistency targets from the conditional velocity $\boldsymbol{\epsilon} - \mathbf{x}_0$, which FlowConsist (Zhang et al., 2026b) shows differs from the marginal velocity by a non-zero covariance term, so the objective acquires a variance penalty whose optimum is not trajectory consistency.

We propose SnapFlow, a self-distillation method that trains a pretrained flow-matching VLA to generate actions in a single action-expert evaluation. SnapFlow mixes standard flow-matching samples, which keep the local velocity field intact, with *shortcut samples* whose target is the average velocity along a two-step Euler path computed by the current model under stop-gradient; the target therefore uses the model's own marginal velocity estimate rather than the conditional velocity, and no teacher copy, EMA, or critic is needed. A single zero-initialized destination-time projection lets the action expert distinguish the two objectives within one architecture, so the pretrained predictor is preserved exactly at initialization and only the action expert and this projection are trained. The VLM backbone stays frozen and its prefix is computed once per batch and shared by the student call and the two target calls, so distillation costs about three action-expert forward passes per example and completes in roughly 12 hours on a single A800 for $\pi0.5$; at deployment nothing is added to the pretrained model except the destination-time input.

Under the archived $\pi0.5$ evaluation, one-NFE SnapFlow reaches 98.75% LIBERO success against 97.75% for the 10-NFE teacher, and under a separate paired evaluation of $\pi0$ it gains 10.5 points over its teacher; these results use separate protocols and are not pooled. SnapFlow also lowers mean and tail action error on held-out data, remains a valid multi-step solver, cuts batch-one model latency by about 3.3×, and transfers to shelf retrieval on a dual-arm humanoid.

Contributions.

- **One-step self-distillation for flow-matching VLAs:** SnapFlow mixes local flow-matching supervision with a model-generated two-step Euler endpoint target, so a pretrained VLA learns direct noise-to-action prediction without a separately trained teacher network, EMA copy, or critic.
- **Destination-time conditioning:** A single zero-initialized projection of the destination time lets one action expert serve both the local-velocity and the endpoint objective; it preserves the pretrained predictor at initialization and is the only architectural change.
- **Evaluation across models and measures:** We report closed-loop LIBERO results for $\pi0.5$ and $\pi0$ under their respective protocols, offline action error and batch-one latency, offline ablations of every training choice, and a qualitative deployment on a dual-arm humanoid.

2 Related Work

Flow-Matching VLAs. $\pi0$ (Black et al., 2024) introduced a flow-matching action head for generalist VLAs, $\pi0.5$ (Physical Intelligence et al., 2025) extends this architecture with open-world training, and SmolVLA (Shukor et al., 2025) provides a lightweight ∼450M alternative. All three generate actions through iterative Euler denoising, typically with 10 sequential action-expert evaluations, which is the cost we target.

VLA Inference Acceleration. One line of work lowers the cost of each of those evaluations. Shallow-π (Jeon et al., 2026) distills a flow-based VLA to fewer transformer layers, EfficientVLA (Yang et al., 2025) prunes language layers and visual tokens while caching action-module features, LightDP (Wu et al., 2025) combines policy compression with consistency distillation for on-device manipulation, and vla.cpp (Nguyen et al., 2026) serves VLAs through a portable runtime with cached-prefix execution. SnapFlow instead reduces the number of evaluations, so the two axes are complementary; Appendix F positions them against each other.

Fast Flow Models. A second line of work, developed for image and video generation, removes sampling steps by changing the training objective. Consistency Models (Song et al., 2023) enforce trajectory self-consistency for single-step generation; in the flow-matching setting, MeanFlow (Geng et al., 2025) models average velocity, ShortCut (Frans et al., 2025) uses two-step target decompositions, α-Flow (Zhang et al., 2026a) anneals from trajectory flow matching toward MeanFlow, and Decoupled MeanFlow (Lee et al., 2026) adapts pretrained flow models through destination-time conditioning. FlowConsist (Zhang et al., 2026b) identifies trajectory drift caused by conditional velocities and proposes a corrected consistency objective, which is the analysis we carry over to action generation in Section 3.3.

Fast Sampling for Robot Policies. These objectives have also been brought to robot policies, though not to pretrained VLAs. Consistency Policy (Prasad et al., 2024) and One-Step Diffusion Policy (Wang et al., 2025) distill small U-Net diffusion policies into few-step generators with an EMA or frozen teacher; FlowPolicy (Zhang et al., 2025), ManiFlow (Yan et al., 2025), and FreqPolicy (Su et al., 2025) train consistency-flow policies from scratch on point clouds, a DiT-X backbone, and frequency-domain constraints, respectively; and Guided Action Flow (Yang et al., 2026) keeps a SmolVLA policy frozen and steers it with a learned critic at test time. SnapFlow differs in that it adapts a pretrained flow-matching VLA through a model-generated Euler target and a single zero-initialized destination-time projection, with no external teacher or critic at deployment.

3 Method

3.1 Preliminaries: Flow Matching in VLAs

Flow-matching VLAs (Black et al., 2024; Physical Intelligence et al., 2025) generate action chunks $\mathbf{x}_0 \in \mathbb{R}^{H\times D}$ by learning a velocity field conditioned on a context $\mathbf{c}$ encoding the observation and language instruction. Given a ground-truth action $\mathbf{x}_0$ and noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, a flow path is defined by linear interpolation:

$$\mathbf{x}_t = (1-t)\,\mathbf{x}_0 + t\,\boldsymbol{\epsilon}, \quad t \in [0,1] \tag{1}$$

The velocity along this path is the *conditional velocity* $\mathbf{v}_t = \boldsymbol{\epsilon} - \mathbf{x}_0$, determined by the specific pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$. Since multiple pairs can produce the same $\mathbf{x}_t$ under a context $\mathbf{c}$, the *marginal velocity field* is defined as $\mathbf{u}_t(\mathbf{x}_t, \mathbf{c}) = \mathbb{E}[\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c}]$. Flow matching trains a network F_θ to approximate $\mathbf{u}_t$ using the conditional velocity as a surrogate:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\big\|F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c}) - (\boldsymbol{\epsilon} - \mathbf{x}_0)\big\|^2\Big] \tag{2}$$

Here $F_\theta(\mathbf{x}_t, s, t \mid \mathbf{c})$ denotes the predicted average velocity from t to target time s; at $s{=}t$ this reduces to the instantaneous velocity.

At inference, the model starts from pure noise $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and integrates backward using K-step Euler:

$$\mathbf{x}_{t-\Delta t} = \mathbf{x}_t - \Delta t \cdot F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c}), \quad \Delta t = 1/K \tag{3}$$

In $\pi0.5$, $K = 10$ is the default, requiring 10 sequential forward passes through the action expert.

3.2 Fast Flow Models and Average Velocity

Rather than using many Euler steps to approximate the ODE integral, a *fast flow model* directly learns the average velocity between time t and a target time $s < t$, enabling a linear

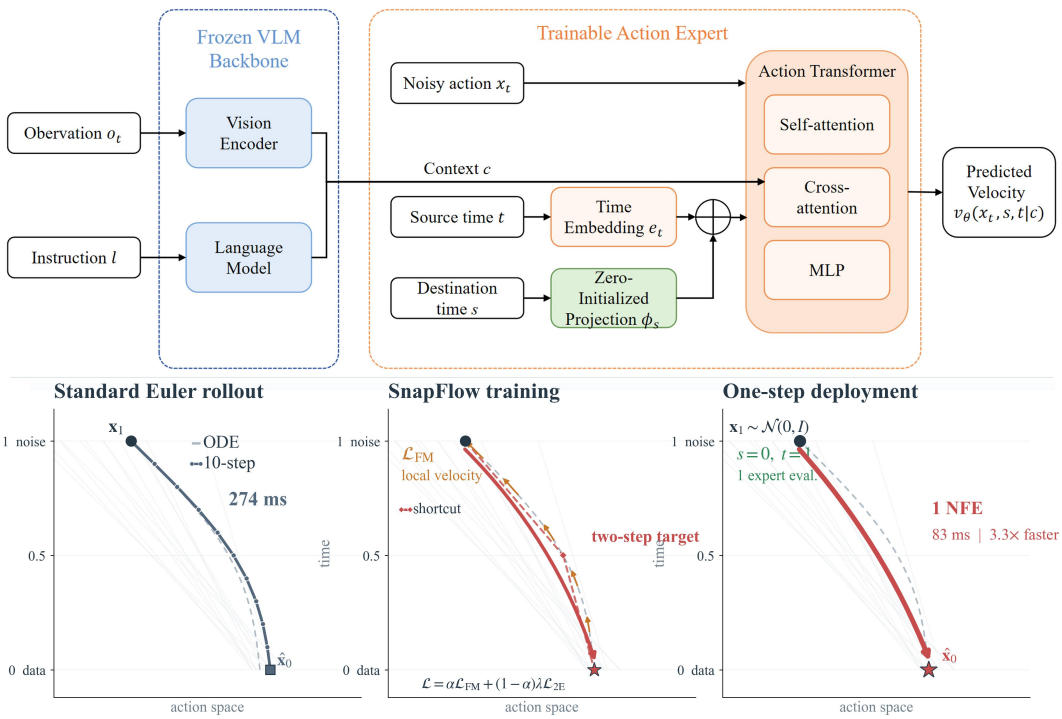

Figure 1: **SnapFlow overview.** Top: the destination-time projection augments the pretrained action expert. (a) The teacher uses 10 Euler evaluations. (b) Training mixes local flow-matching supervision with a model-generated two-step endpoint target. (c) Destination $s = 0$ produces an action chunk with one expert evaluation.

mapping (Geng et al., 2025; Zhang et al., 2026a):

$$f_\theta(\mathbf{x}_t, s, t) = \mathbf{x}_t - (t - s)\, F_\theta(\mathbf{x}_t, s, t \mid \mathbf{c}) \tag{4}$$

where $F_\theta(\mathbf{x}_t, s, t)$ approximates the true average velocity $\mathbf{u}_{\text{avg}}(\mathbf{x}_t, s, t) = \frac{1}{t-s}\int_s^t \mathbf{u}(\mathbf{x}_\tau, \tau)\, d\tau$. Setting $s = 0$ and $t = 1$ yields the desired 1-NFE mapping from noise to action: $\hat{\mathbf{x}}_0 = \mathbf{x}_1 - F_\theta(\mathbf{x}_1, 0, 1)$.

The trajectory consistency objective (Song et al., 2023; Geng et al., 2025; Zhang et al., 2026b) enforces that the predicted endpoint $f_\theta(\mathbf{x}_t, s, t)$ is invariant to the starting time t along the same trajectory:

$$\mathbb{E}_{\mathbf{x}_t}\left[\left\|\frac{d}{dt} f_\theta(\mathbf{x}_t, s, t)\right\|^2\right] = \mathbb{E}_{\mathbf{x}_t}\left[\left\|\nabla_{\mathbf{x}_t} f_\theta \cdot \mathbf{u}_t + \partial_t f_\theta\right\|^2\right] = 0 \tag{5}$$

In practice $\mathbf{u}_t$ is unknown and is typically replaced by the conditional velocity $\mathbf{v}_t = \boldsymbol{\epsilon} - \mathbf{x}_0$. For standard flow matching at $s = t$, this substitution is valid because $\mathbb{E}[\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c}] = \mathbf{u}_t$. However, for fast flow models that require trajectory consistency across a finite time span $s \neq t$, we show below that this substitution introduces systematic drift.

3.3 Trajectory Consistency Analysis

We adapt three FlowConsist results to the context-conditioned VLA setting; complete proofs are in Appendix A.

Theorem 1 (Conditional–Marginal Velocity Discrepancy (Zhang et al., 2026b)). Let $\mathbf{x}_0 \sim p_{\text{data}}$ be non-degenerate (not a Dirac mass) and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. For contexts where $p(\mathbf{x}_0 \mid \mathbf{c})$ is non-degenerate, let $\mathbf{u}_t = \mathbb{E}[\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c}]$. The conditional covariance $\boldsymbol{\Sigma}_t(\mathbf{x}_t, \mathbf{c}) = \text{Cov}(\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c})$ is generally non-zero.

Proof sketch. At $t{=}0$, $\mathbf{x}_t = \mathbf{x}_0$ is deterministic given the data, so $\mathbf{v}_0 = \boldsymbol{\epsilon} - \mathbf{x}_0$ has variance $\mathrm{Var}(\boldsymbol{\epsilon}) = \mathbf{I}$. For $t \in (0, 1]$, conditioning throughout on $\mathbf{c}$ and substituting $\boldsymbol{\epsilon} = (\mathbf{x}_t - (1{-}t)\mathbf{x}_0)/t$ gives $\boldsymbol{\Sigma}_t = t^{-2}\,\mathrm{Var}(\mathbf{x}_0 \mid \mathbf{x}_t, \mathbf{c})$. Under the stated non-degeneracy assumption, this posterior covariance is generally non-zero. □

Theorem 2 (Conditional-Objective Decomposition (Zhang et al., 2026b))**.** Let the conditional training objective be $\mathcal{L}_{\mathrm{cond}}(\theta) = \mathbb{E}_{\mathbf{x}_t,\mathbf{v}_t}[\|\nabla_{\mathbf{x}_t} f_\theta \cdot \mathbf{v}_t + \partial_t f_\theta\|^2]$. Then:

$$\mathcal{L}_{\mathrm{cond}}(\theta) = \underbrace{\mathbb{E}_{\mathbf{x}_t}\left[\|\nabla_{\mathbf{x}_t} f_\theta \cdot \mathbf{u}_t + \partial_t f_\theta\|^2\right]}_{\mathcal{L}_{\mathrm{consist}}(\theta)} + \underbrace{\mathbb{E}_{\mathbf{x}_t}\left[\mathrm{Tr}\big(\nabla_{\mathbf{x}_t} f_\theta\, \boldsymbol{\Sigma}_t(\mathbf{x}_t)\, (\nabla_{\mathbf{x}_t} f_\theta)^\top\big)\right]}_{\mathcal{L}_{\mathrm{var}}(\theta)} \tag{6}$$

The two objectives differ whenever $\mathrm{Tr}(\nabla_{\mathbf{x}_t} f_\theta\, \boldsymbol{\Sigma}_t\, (\nabla_{\mathbf{x}_t} f_\theta)^\top) > 0$.

Proof sketch. Decompose $\mathbf{v}_t = \mathbf{u}_t + (\mathbf{v}_t - \mathbf{u}_t)$ in the quadratic loss. The cross term vanishes because $\mathbb{E}[\mathbf{v}_t - \mathbf{u}_t \mid \mathbf{x}_t, \mathbf{c}] = \mathbf{0}$. The residual is $\mathcal{L}_{\mathrm{var}}$, a positive-semidefinite quadratic form in $\nabla_{\mathbf{x}_t} f_\theta$ weighted by $\boldsymbol{\Sigma}_t$. □

Remark 1. The variance term $\mathcal{L}_{\mathrm{var}}$ penalizes components of $\nabla_{\mathbf{x}_t} f_\theta$ in high-variance directions. For standard flow matching ($s{=}t$), this does not matter because $f_\theta(\mathbf{x}_t, t, t) = \mathbf{x}_t$ is trivially consistent. For finite-span maps ($s \neq t$), the conditional and marginal consistency objectives can therefore have different optima.

Theorem 3 (Cumulative Error in Consistency Mapping (Zhang et al., 2026b))**.** Let $f^*(\mathbf{x}_t, s, t)$ denote the ideal consistency mapping and $f_\theta(\mathbf{x}_t, s, t)$ the learned model. Define the local residual $R(t) = \partial_t f_\theta + \nabla_{\mathbf{x}_t} f_\theta \cdot \mathbf{u}_t$ and the total error $e(s, t) = f_\theta(\mathbf{x}_t, s, t) - f^*(\mathbf{x}_t, s, t)$. Then:

$$e(s, t) = \int_s^t R(r)\, dr \tag{7}$$

Consequently, $\|e(s, t)\| \leq \int_s^t \|R(r)\|\, dr$: longer mappings expose the model to accumulated local residuals, although cancellation can prevent monotonic growth.

Remark 2. Theorem 3 motivates learning a direct endpoint mapping rather than repeatedly applying imperfect local predictions. Section 4.3 evaluates this design empirically.

3.4 SnapFlow: Corrected Consistency Training for VLAs

Motivated by Theorems 1–3, SnapFlow replaces the conditional velocity in the consistency target with the model's own marginal velocity prediction and stochastically mixes consistency and standard FM supervision. We develop the objective in three steps: we first correct the consistency loss, then replace its expensive total-derivative term with a two-step Euler target, and finally combine the result with flow matching.

The correction follows Zhang et al. (2026b): replacing $\mathbf{u}_t$ with $\mathbf{v}_t$ in the first term of the consistency loss introduces only a parameter-free constant (Appendix A.5), whereas the total-derivative term must use the marginal estimate $\mathbf{u}_\theta = F_\theta(\mathbf{x}_t, t, t)$ identified in Theorem 2. This yields:

$$\mathcal{L}_{\mathrm{consist}} = \mathbb{E}_{\mathbf{x}_t}\left[\|F_\theta(\mathbf{x}_t, s, t) - \mathrm{sg}(\mathbf{v}_t - (t - s)(\nabla_{\mathbf{x}_t} F_\theta \cdot \mathbf{u}_\theta + \partial_t F_\theta))\|^2\right] \tag{8}$$

where $\mathrm{sg}(\cdot)$ denotes stop-gradient and $\mathbf{u}_\theta = F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c})$ is the model's marginal velocity estimate, maintained by the FM component of training.

Computing the total derivative $\nabla_{\mathbf{x}_t} F_\theta \cdot \mathbf{u}_\theta + \partial_t F_\theta$ in Eq. (8) is expensive for billion-parameter VLAs, so we approximate it with two equal stop-gradient Euler steps generated by the current VLA. This finite-step target is derived for SnapFlow's corrected consistency objective and uses the model's own marginal velocity estimates; it shares the general two-half-step structure of Shortcut Models (Frans et al., 2025). For a noisy action $\mathbf{x}_t$ at a sampled source time t, the current model takes two equal stop-gradient Euler steps:

$$\mathbf{v}^{(1)} = \mathrm{sg}(F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c}))\,, \qquad \mathbf{x}_{t/2} = \mathbf{x}_t - \frac{t}{2}\mathbf{v}^{(1)}, \tag{9}$$

$$\mathbf{v}^{(2)} = \mathrm{sg}\big(F_\theta(\mathbf{x}_{t/2}, t/2, t/2 \mid \mathbf{c})\big)\,, \qquad \widehat{\mathbf{x}}_0^{(2)} = \mathbf{x}_{t/2} - \frac{t}{2}\mathbf{v}^{(2)}. \tag{10}$$

The shortcut target is the one-step velocity that reaches this two-step endpoint:

$$\mathbf{v}_{\mathrm{2E}} = \mathrm{clip}\left(\frac{\mathbf{x}_t - \widehat{\mathbf{x}}_0^{(2)}}{t}, -20, 20\right) = \mathrm{clip}\left(\frac{\mathbf{v}^{(1)} + \mathbf{v}^{(2)}}{2}, -20, 20\right). \tag{11}$$

This target is the average displacement induced by two explicit-Euler steps, and it is built entirely from the model's own marginal velocity estimates, as the correction requires; as the local velocity field improves under FM supervision, the endpoint target improves with it.

The shortcut target is only as good as the local field that generates it, so the final objective keeps training that field. Each example uses destination $s = t$ with probability α and $s = 0$ otherwise, and the sampled target and weight are

$$(\mathbf{y}, w) = \begin{cases} (\boldsymbol{\epsilon} - \mathbf{x}_0, 1), & s = t, \\ (\mathbf{v}_{\mathrm{2E}}, \lambda), & s = 0, \end{cases} \qquad \mathcal{L} = \mathbb{E}\Big[w \, \|F_\theta(\mathbf{x}_t, s, t \mid \mathbf{c}) - \mathbf{y}\|_2^2\Big]. \tag{12}$$

In expectation, this gives $\alpha\mathcal{L}_{\mathrm{FM}} + (1-\alpha)\lambda\mathcal{L}_{\mathrm{2E}}$. The FM branch maintains the local velocity estimator, while the shortcut branch teaches direct endpoint prediction.

3.5 Destination-Time Conditioning

Flow-map models condition predictions on a destination time (Lee et al., 2026). To distinguish FM ($s = t$) from endpoint ($s = 0$) samples, we encode s with the source-time sinusoidal basis and pass it through a zero-initialized linear projection ϕ_s. The projected destination embedding is added to the action expert's time-conditioning pathway. Zero initialization preserves the pretrained predictor at step 0 and introduces only a small model-specific modification.

3.6 Training and Inference

SnapFlow freezes the VLM backbone and trains the action expert plus the destination-time projection (approximately 300M trainable expert parameters in the archived $\pi0.5$ setup) with gradient checkpointing. Appendix K reports the optimization settings and Appendix J the early-training stability of the self-referential target. At deployment (Figure 1c), a single forward pass produces the action chunk:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_1 - F_\theta(\mathbf{x}_1, s{=}0, t{=}1 \mid \mathbf{c}), \quad \mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{13}$$

4 Experiments

4.1 Experimental Setup

We apply SnapFlow to two flow-matching VLAs evaluated under distinct configurations: $\pi0.5$ (Physical Intelligence et al., 2025), whose archived 10-NFE teacher reaches 97.75% success, and $\pi0$ (Black et al., 2024), whose teacher reaches 70.50% under the configuration used here; training schedules are listed in Appendix K. SmolVLA (Shukor et al., 2025), a ∼450M VLA with an interleaved self-/cross-attention expert, provides an offline cross-architecture comparison on PushT.

Both closed-loop studies use the four LIBERO suites (Liu et al., 2023), 10 tasks each and 10 episodes per task (400 trials per method), but they differ in protocol. The archived $\pi0.5$ study follows the original aggregate evaluation. The $\pi0$ study evaluates the three methods in a paired manner: the 10 episodes of each task start from the same 10 LIBERO initial states, shared across teacher, naïve one-step, and SnapFlow, under a common evaluation seed; each policy predicts a 50-step action chunk and executes the first 10 steps before the next inference. The two studies were run at different times with different training runs, so their numbers are reported separately and are not pooled. Offline metrics use 500 held-out samples, and latency is synchronized batch-one inference on one A800-80G.

In every comparison we report three methods: the teacher, meaning the pretrained model with its default 10-step Euler integration; naïve one-step, meaning the same model evaluated with a single step and no retraining; and SnapFlow, meaning the distilled model evaluated with a single step.

Table 1: **LIBERO evaluation under two configurations.** The $\pi 0.5$ rows reproduce the archived aggregate evaluation; the $\pi 0$ rows come from a separate paired evaluation under the configuration used here and are retained as a secondary result. SnapFlow rows use the default training configuration; Section 4.4 evaluates offline training variants. Latency is synchronized batch-one model inference on one A800-80G; speedups are relative to each model's own 10-NFE teacher. †Reference scores compiled in Appendix E of OpenVLA (Kim et al., 2024) under per-suite fine-tuning, listed for context only.

		LIBERO Success (%)					**Latency (A800)**	
Method	NFE	Spatial	Object	Goal	Long-10	Avg	Model↓	Model Speedup
Published references†								
Diff. Policy (Chi et al., 2023)	100	78.3	92.5	68.3	50.5	72.40	—	
Octo-Base (Ghosh et al., 2024)	10	78.9	85.7	84.6	51.1	75.08	—	
OpenVLA (Kim et al., 2024)	AR	84.7	88.4	79.2	53.7	76.50	—	
Published $\pi 0$ (Black et al., 2024)	10	97.4	98.4	97.6	93.0	96.60	—	
$\pi 0.5$ (Physical Intelligence et al., 2025)		*archived teacher (97.75%)*						
Teacher	10	98.0	**100.0**	96.0	97.0	97.75	274 ms	1.00×
Naïve 1-step	1	96.0	99.0	98.0	94.0	96.75	**81 ms**	**3.4**×
SnapFlow	1	**99.0**	**100.0**	**99.0**	**97.0**	**98.75**	83 ms	3.3×
$\pi 0$ (Black et al., 2024)		*in-configuration teacher (70.50%)*						
Teacher	10	71.0	85.0	82.0	44.0	70.50	277.9 ms	1.00×
Naïve 1-step	1	87.0	85.0	**89.0**	50.0	77.75	**82.8 ms**	**3.36**×
SnapFlow	1	**88.0**	**87.0**	**89.0**	**60.0**	**81.00**	83.0 ms	3.35×

4.2 Main Results

Table 1 reports the two model results side by side; the $\pi 0.5$ rows use the archived aggregate protocol, whereas the $\pi 0$ rows use a separate paired protocol tied to the configuration used here.

On $\pi 0.5$, under the archived aggregate evaluation, SnapFlow reaches 98.75% average success, ahead of the 97.75% 10-NFE teacher and the 96.75% naïve one-step baseline, and is best on each of the four suites; per-task outcomes are listed in Appendix C.

On $\pi 0$, we report the paired evaluation as a secondary, within-configuration comparison. The teacher score of 70.50% is substantially below the published $\pi 0$ reference of 96.60%; repeated attempts under our setup did not reproduce that reference, and we have not isolated the cause of the gap. We therefore do not present this run as a reproduction of the published benchmark result. On matched initial states, naïve one-step and SnapFlow reach 77.75% and 81.00%, respectively: SnapFlow's gain is 3.25 percentage points over the undistilled one-step baseline and 10.5 points over this run's teacher. The largest gain over naïve one-step is on the long-horizon suite (60.0% vs. 50.0%).

4.3 Quality–Latency Trade-off

Success rate alone does not show whether one-step generation changes the predicted actions themselves, so Figure 2 and Table 2 examine held-out action error alongside closed-loop success; the latency decomposition appears in Appendix G.

On $\pi 0.5$, SnapFlow lowers mean MSE by 33.9%, median MSE by 7.6%, P90 by 23.6%, and P95 by 29.4%, its standard deviation falls by 45.2%, and CosSim increases from .9885 to .9916, so the improvement is not confined to the mean but extends to the tail of the error distribution. Offline error and closed-loop success remain distinct quantities, but the archived result improves both aggregate success and held-out action metrics. The distilled model also

Table 2: **Extended offline metrics.** $\pi 0.5$ uses the archived 500-sample LIBERO evaluation; SmolVLA uses 500 PushT samples.

Method	Avg MSE↓	Med MSE↓	Std MSE↓	P90 MSE↓	P95 MSE↓	CosSim↑
$\pi 0.5$ — LIBERO (500 samples)						
Baseline (10-step)	.01169	.00397	.05412	.01544	.02357	.9885
SnapFlow (1-step)	**.00773**	**.00367**	**.02964**	**.01179**	**.01664**	**.9916**
Relative change: mean −33.9%, median −7.6%, std. −45.2%, P90 −23.6%, P95 −29.4%						
SmolVLA — PushT						
Baseline (10-step)	0.468	0.268	0.517	1.162	—	0.765
SnapFlow (1-step)	**0.429**	0.272	**0.452**	**1.029**	—	**0.818**
Δ: MSE −8.3%, Std −12.6%, P90 −11.4%, CosSim +6.9%						

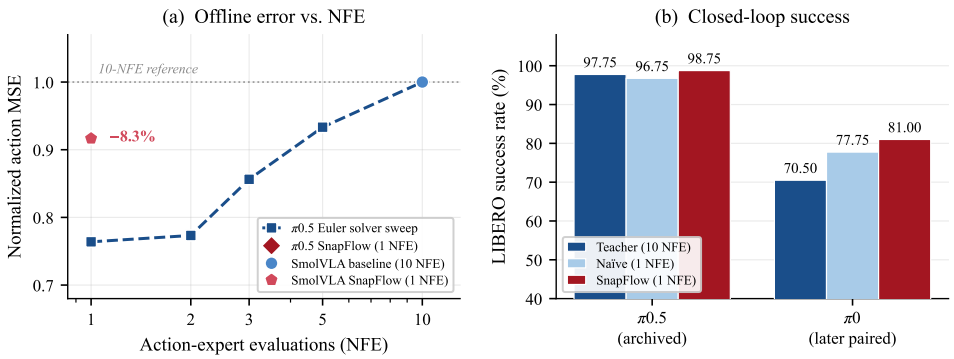

Figure 2: **Quality frontier across VLAs and closed-loop success.** (a) Offline action MSE against the number of action-expert evaluations, normalized within each model by its 10-NFE reference. Truncating the pretrained $\pi 0.5$ solver traces the dashed frontier; one-NFE SnapFlow lands below the 10-NFE reference on the archived $\pi 0.5$ and SmolVLA offline sets. (b) LIBERO success for the archived $\pi 0.5$ evaluation and the separate $\pi 0$ evaluation.

remains a valid multi-step solver: Appendix H sweeps its step count and finds lower error than the teacher at every NFE.

4.4 Ablation Studies

Table 3 varies each of SnapFlow's three training choices, the mixing ratio α, the shortcut weight λ, and destination-time conditioning, on the archived $\pi 0.5$ held-out set; these are offline diagnostics, not additional closed-loop claims.

In each block the lowest MSE occurs at the default setting, $\alpha = 0.5$ and $\lambda = 0.1$ with destination-time conditioning; both extremes of α are worse, which shows that the FM and shortcut branches are both needed, and removing the conditioning raises MSE from .0077 to .0098 because both branches then receive the same time signal.

4.5 Real-Robot Deployment

To check that one-step generation survives the move from simulation to physical hardware, we fine-tuned the same $\pi 0.5$ backbone on teleoperated demonstrations collected on a dual-arm AgiBot humanoid, distilled it with SnapFlow under the default $\alpha = 0.5$ and $\lambda = 0.1$, and deployed the resulting policy with a single action-expert evaluation. The task is shelf retrieval in a retail pharmacy: the robot reaches into a stocked shelf, closes the gripper on a designated product, and removes it. Each shelf configuration has its own target product and its own

Table 3: **Offline ablation study** ($\pi0.5$, 500 held-out samples). Each block changes one training choice; otherwise $\alpha=0.5$, $\lambda=0.1$, and destination-time conditioning is enabled.

Variant	Configuration α	Configuration λ	Configuration Dest.-time conditioning	One-NFE quality MSE↓	One-NFE quality CosSim↑	Observation
(a) FM/Consistency Mixing Ratio α						
Pure consistency	0.0	0.1	✓	.0115	.9876	No FM branch
Consistency-heavy	0.3	0.1	✓	.0088	.9901	70% endpoint samples
Balanced (default)	0.5	0.1	✓	**.0077**	**.9916**	Equal branch probability
FM-heavy	0.7	0.1	✓	.0084	.9908	30% endpoint samples
Pure FM	1.0	0.1	✓	.0093	.9896	No endpoint branch
(b) Consistency Weight λ						
Low weight	0.5	0.01	✓	.0089	.9902	$\lambda = 0.01$
Default	0.5	0.1	✓	**.0077**	**.9916**	$\lambda = 0.1$
High weight	0.5	1.0	✓	.0096	.9891	$\lambda = 1.0$
(c) Destination-Time Conditioning						
Without conditioning	0.5	0.1	×	.0098	.9889	Same time signal for both objectives
With conditioning (default)	0.5	0.1	✓	**.0077**	**.9916**	Separate destination signal



Figure 3: **Real-robot shelf retrieval on a dual-arm AgiBot humanoid.** Keyframes at matched fractions of each episode for the 10-NFE $\pi0.5$ teacher and for one-NFE SnapFlow on the same shelf configuration. Both policies reach the shelf, grasp the target product, and extract it. The recordings are a deployment demonstration and are not used for success-rate or latency claims.

teacher–SnapFlow pair trained on demonstrations for that target; Appendix L describes the setup and shows the remaining configurations.

Figure 3 shows one shelf configuration executed by both policies; both reach into the shelf, grasp the target product, and extract it, so a single destination-time evaluation still yields action chunks that a physical arm can execute on contact-rich shelf geometry. The recordings were made after fine-tuning had converged on each target, and neither policy failed to grasp in these sessions; however, trial counts and initial states were not fixed in advance, and nine episodes across four configurations are too few for a success-rate comparison. We therefore treat the real-robot material as a qualitative transfer check, keep every quantitative closed-loop claim in the LIBERO evaluations of Section 4.2, and report no real-robot latency, which was not measured under the protocol of Appendix G.

5 Conclusion

SnapFlow adapts a pretrained flow-matching VLA to one-NFE action generation using model-generated Euler targets and destination-time conditioning. It reaches 98.75% success on archived $\pi0.5$ versus 97.75% for its teacher with lower held-out action error, improves our $\pi0$ teacher by 10.5 points, and cuts batch-one model latency by about 3.3×.

Quantitative closed-loop evidence is limited to two PaliGemma-based VLAs on LIBERO; real-robot evidence is qualitative. SnapFlow requires a flow-matching checkpoint. The shared VLM prefix now dominates model latency, motivating complementary layer distillation (Jeon et al., 2026) and token pruning (Yang et al., 2025).

AI Use Statement

Generative AI tools were used for manuscript polishing and typesetting, organizing code and experimental results, and code standardization. The authors take responsibility for the final manuscript.

Ethics Statement

This work studies inference-time efficiency of robot manipulation policies. The simulation experiments use only publicly available datasets and pretrained checkpoints (LIBERO (Liu et al., 2023), PushT, $\pi 0$ (Black et al., 2024), $\pi 0.5$ (Physical Intelligence et al., 2025), SmolVLA (Shukor et al., 2025)). The real-robot demonstration of Section 4.5 uses teleoperated manipulation data that we collected on our own hardware in a shelf environment stocked with commercial retail packaging; it involves no human subjects, no personal data, and no third-party data collection. Reducing action-generation latency lowers the compute cost of deploying manipulation policies, but it does not alter what tasks a policy is capable of, and the safety considerations of the underlying pretrained VLAs remain unchanged. Physical deployment adds the standard requirements of manipulator operation: all recordings were made in a closed test area with an operator able to halt the robot.

Reproducibility Statement

Section 4.1 specifies the models, benchmarks, and their separate evaluation protocols, and Appendix K lists the complete training configuration. The training objective and one-NFE inference path are given as pseudocode in Appendix B, and the derivations behind the objective, together with their assumptions, are proved in Appendix A. Per-task outcomes for every reported closed-loop number appear in Appendix C, so aggregate values can be recomputed from the tables. The supplementary material contains the complete real-robot recordings behind Figure 3 and Appendix L, together with synchronized side-by-side LIBERO rollouts of the 10-NFE teacher and one-NFE SnapFlow; Appendix M indexes the files.

References

K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *RSS*, 2023.

K. Frans, D. Hafner, S. Levine, and P. Abbeel. One step diffusion via shortcut models. In *ICLR*, 2025.

Z. Geng, M. Deng, X. Bai, J. Z. Kolter, and K. He. Mean flows for one-step generative modeling. In *NeurIPS*, 2025.

D. Ghosh, H. R. Walke, K. Pertsch, K. Black, O. Mees, et al. Octo: An open-source generalist robot policy. In *RSS*, 2024.

B. Jeon, Y. Choi, and T. Kim. Shallow-π: Knowledge distillation for flow-based VLAs. *arXiv preprint arXiv:2601.20262*, 2026.

M. J. Kim, K. Pertsch, S. Karamcheti, et al. OpenVLA: An open-source vision-language-action model. In *CoRL*, 2024.

K. Lee, S. Yu, and J. Shin. Decoupled MeanFlow: Turning flow models into flow maps for accelerated sampling. In *ICLR*, 2026.

Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *ICLR*, 2023.

B. Liu, Y. Zhu, C. Gao, Y. Feng, et al. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS Datasets and Benchmarks Track*, 2023.

K. D. Nguyen, H. T. Ho, C. T. Nguyen, T. Q. Duong, L. D. Le, et al. vla.cpp: A unified inference runtime for vision-language-action models. *arXiv preprint arXiv:2606.08094*, 2026.

Physical Intelligence, K. Black, N. Brown, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

A. Prasad, K. Lin, J. Wu, L. Zhou, and J. Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. In *RSS*, 2024.

M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, et al. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. Consistency models. In *ICML*, 2023.

Y. Su, N. Liu, D. Chen, Z. Zhao, K. Wu, M. Li, Z. Xu, Z. Che, and J. Tang. FreqPolicy: Efficient flow-based visuomotor policy via frequency consistency. In *NeurIPS*, 2025.

Z. Wang, M. Li, A. Mandlekar, Z. Xu, J. Fan, Y. Narang, et al. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. In *ICML*, 2025.

Y. Wu, H. Wang, Z. Chen, J. Pang, and D. Xu. On-device diffusion transformer policy for efficient robot manipulation. In *ICCV*, 2025.

G. Yan, J. Zhu, Y. Deng, S. Yang, R.-Z. Qiu, X. Cheng, et al. ManiFlow: A general robot manipulation policy via consistency flow training. In *CoRL*, 2025.

Y. Yang, Y. Wang, Z. Wen, Z. Luo, C. Zou, Z. Zhang, C. Wen, and L. Zhang. EfficientVLA: Training-free acceleration and compression for vision-language-action models. In *NeurIPS*, 2025.

L. Yang, Z. Jiang, C. Sheng, and Z. Tang. Guided action flow: Q-guided inference for flow-matching vision-language-action policies. *arXiv preprint arXiv:2607.02092*, 2026.

H. Zhang, A. Siarohin, W. Menapace, M. Vasilkovsky, S. Tulyakov, Q. Qu, and I. Skorokhodov. AlphaFlow: Understanding and improving MeanFlow models. In *ICLR*, 2026.

Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu. FlowPolicy: Enabling fast and robust 3D flow-based policy via consistency flow matching for robot manipulation. In *AAAI*, 2025.

T. Zhang, C. Liu, J. Chen, C.-L. Guo, C. Li, M.-M. Cheng, B. Li, and P.-T. Jiang. FlowConsist: Make your flow consistent with real trajectory. *arXiv preprint arXiv:2602.06346*, 2026.

A Notation and Theoretical Proofs

This appendix supports the analysis of Section 3.3: it fixes the notation used throughout the method section, gives complete proofs of the three theorems that were only sketched there, and derives the corrected objective of Eq. (8).

A.1 Notation

Because the method section introduces several closely related velocities and time variables, we first collect every symbol in one place, as listed in Table 4.

The remaining subsections prove the three theorems stated in Section 3.3.

Table 4: **Notation.** Times run from $t=1$ (noise) to $t=0$ (clean action), so integration proceeds backward in t and the destination time satisfies $s \le t$.

Symbol	Meaning
$\mathbf{x}_0 \in \mathbb{R}^{H\times D}$	Action chunk of horizon H and action dimension D
$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$	Gaussian noise sample
$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\boldsymbol{\epsilon}$	Point on the linear flow path at source time t
$\mathbf{c}$	Context: VLM prefix encoding the observation and instruction
$t \in [0,1]$	Source time; $t=1$ is noise, $t=0$ is the clean action
$s \in [0,t]$	Destination time; $s=t$ gives flow matching, $s=0$ gives one-step generation
$\mathbf{v}_t = \boldsymbol{\epsilon} - \mathbf{x}_0$	Conditional velocity of the sampled pair $(\mathbf{x}_0, \boldsymbol{\epsilon})$
$\mathbf{u}_t(\mathbf{x}_t, \mathbf{c}) = \mathbb{E}[\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c}]$	Marginal (context-conditioned) velocity field
$\boldsymbol{\Sigma}_t(\mathbf{x}_t, \mathbf{c}) = \mathrm{Cov}(\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c})$	Conditional velocity covariance
$F_\theta(\mathbf{x}_t, s, t \mid \mathbf{c})$	Network output: average velocity from t to s
$\mathbf{u}_\theta = F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c})$	Model estimate of the marginal velocity
$f_\theta(\mathbf{x}_t, s, t) = \mathbf{x}_t - (t-s)F_\theta$	Induced flow map
$\mathbf{u}_{\mathrm{avg}}(\mathbf{x}_t, s, t)$	True average velocity $\frac{1}{t-s}\int_s^t \mathbf{u}(\mathbf{x}_\tau, \tau)\, d\tau$
$\mathbf{v}_{\mathrm{2E}}$	Two-step Euler shortcut target (Eq. 11)
α	Probability of drawing a flow-matching sample ($s=t$)
λ	Loss weight on shortcut samples ($s=0$)
ϕ_s	Zero-initialized destination-time projection
$\mathrm{sg}(\cdot)$	Stop-gradient operator
K	Number of Euler steps at inference; NFE counts action-expert evaluations
n_{act}	Action-chunk steps executed before the next inference

A.2 Proof of Theorem 1 (Conditional–Marginal Velocity Discrepancy)

Proof. We analyze two cases.

Case 1: $t = 0$**.** At $t=0$, $\mathbf{x}_t = \mathbf{x}_0$. The conditional velocity is $\mathbf{v}_0 = \boldsymbol{\epsilon} - \mathbf{x}_0$. Since $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is independent of $(\mathbf{x}_0, \mathbf{c})$, the marginal velocity is $\mathbf{u}_0(\mathbf{x}_0, \mathbf{c}) = \mathbb{E}[\boldsymbol{\epsilon} - \mathbf{x}_0 \mid \mathbf{x}_0, \mathbf{c}] = -\mathbf{x}_0$. Therefore:

$$\boldsymbol{\Sigma}_0(\mathbf{x}_0, \mathbf{c}) = \mathrm{Var}(\mathbf{v}_0 \mid \mathbf{x}_0, \mathbf{c}) = \mathrm{Var}(\boldsymbol{\epsilon}) = \mathbf{I} \neq \mathbf{0} \tag{14}$$

Case 2: $t \in (0, 1]$**.** Substituting $\boldsymbol{\epsilon} = (\mathbf{x}_t - (1-t)\mathbf{x}_0)/t$ into $\mathbf{v}_t = \boldsymbol{\epsilon} - \mathbf{x}_0$:

$$\mathbf{v}_t = \frac{\mathbf{x}_t - (1-t)\mathbf{x}_0}{t} - \mathbf{x}_0 = \frac{1}{t}(\mathbf{x}_t - \mathbf{x}_0) \tag{15}$$

The conditional covariance is therefore:

$$\boldsymbol{\Sigma}_t(\mathbf{x}_t, \mathbf{c}) = \mathrm{Var}(\mathbf{v}_t \mid \mathbf{x}_t, \mathbf{c}) = \frac{1}{t^2}\,\mathrm{Var}(\mathbf{x}_0 \mid \mathbf{x}_t, \mathbf{c}) \tag{16}$$

The posterior is $p(\mathbf{x}_0 \mid \mathbf{x}_t, \mathbf{c}) \propto p(\mathbf{x}_0 \mid \mathbf{c})\,\mathcal{N}\Big(\frac{\mathbf{x}_t-(1-t)\mathbf{x}_0}{t}; \mathbf{0}, \mathbf{I}\Big)$. For contexts with a non-degenerate action distribution, this posterior generally retains non-zero variance. Therefore $\boldsymbol{\Sigma}_t(\mathbf{x}_t, \mathbf{c})$ is generally non-zero. □

A.3 Proof of Theorem 2 (Conditional-Objective Decomposition)

Proof. Let $J_\theta = \nabla_{\mathbf{x}_t} f_\theta$ and $\dot{f}_\theta = \partial_t f_\theta$. The conditional objective can be written as:

$$\mathcal{L}_{\text{cond}}(\theta) = \mathbb{E}_{\mathbf{x}_t,\mathbf{c}}\Big[\mathbb{E}_{\mathbf{v}_t|\mathbf{x}_t,\mathbf{c}}\Big[\|J_\theta \mathbf{v}_t + \dot{f}_\theta\|^2\Big]\Big] \tag{17}$$

Decompose $\mathbf{v}_t = \mathbf{u}_t + \boldsymbol{\delta}_t$ where $\boldsymbol{\delta}_t = \mathbf{v}_t - \mathbf{u}_t$ and $\mathbb{E}[\boldsymbol{\delta}_t \mid \mathbf{x}_t, \mathbf{c}] = \mathbf{0}$:

$$\begin{aligned}\|J_\theta \mathbf{v}_t + \dot{f}_\theta\|^2 &= \|(J_\theta \mathbf{u}_t + \dot{f}_\theta) + J_\theta \boldsymbol{\delta}_t\|^2 \\ &= \|J_\theta \mathbf{u}_t + \dot{f}_\theta\|^2 + 2(J_\theta \mathbf{u}_t + \dot{f}_\theta)^\top J_\theta \boldsymbol{\delta}_t + \|J_\theta \boldsymbol{\delta}_t\|^2\end{aligned} \tag{18}$$

We now take the conditional expectation $\mathbb{E}_{\mathbf{v}_t|\mathbf{x}_t,\mathbf{c}}[\cdot]$ of each of the three terms. The first term, $\|J_\theta \mathbf{u}_t + \dot{f}_\theta\|^2$, does not depend on $\mathbf{v}_t$ and is therefore unchanged by the expectation. The cross term vanishes, because J_θ and $\dot{f}_\theta$ are deterministic given $\mathbf{x}_t$ while $\boldsymbol{\delta}_t$ has zero conditional mean: $\mathbb{E}[J_\theta \boldsymbol{\delta}_t \mid \mathbf{x}_t, \mathbf{c}] = J_\theta\, \mathbb{E}[\boldsymbol{\delta}_t \mid \mathbf{x}_t, \mathbf{c}] = \mathbf{0}$. The third term is the expected squared norm of a linear image of $\boldsymbol{\delta}_t$, which equals a trace against its covariance: $\mathbb{E}[\|J_\theta \boldsymbol{\delta}_t\|^2 \mid \mathbf{x}_t, \mathbf{c}] = \mathrm{Tr}(J_\theta\, \boldsymbol{\Sigma}_t(\mathbf{x}_t, \mathbf{c})\, J_\theta^\top)$. Taking the outer expectation over $\mathbf{x}_t$ completes the decomposition:

$$\mathcal{L}_{\text{cond}}(\theta) = \underbrace{\mathbb{E}_{\mathbf{x}_t}[\|J_\theta \mathbf{u}_t + \dot{f}_\theta\|^2]}_{\mathcal{L}_{\text{consist}}} + \underbrace{\mathbb{E}_{\mathbf{x}_t}[\mathrm{Tr}(J_\theta\, \boldsymbol{\Sigma}_t\, J_\theta^\top)]}_{\mathcal{L}_{\text{var}}} \tag{19}$$

The variance contribution is positive semidefinite and is strictly positive whenever J_θ has a component in a non-zero covariance direction. □

A.4 Proof of Theorem 3 (Cumulative Error)

Proof. Let $\{\mathbf{x}_r\}_{r\in[s,t]}$ follow the marginal flow: $\frac{d\mathbf{x}_r}{dr} = \mathbf{u}_r(\mathbf{x}_r)$. The ideal consistency mapping satisfies $f^*(\mathbf{x}_t, s, t) = \mathbf{x}_s$ for all t, hence its total derivative along the flow vanishes:

$$\frac{d}{dt} f^*(\mathbf{x}_t, s, t) = \partial_t f^* + \nabla_{\mathbf{x}_t} f^* \cdot \mathbf{u}_t = 0 \tag{20}$$

The total derivative of the error $e(s,t) = f_\theta(\mathbf{x}_t, s, t) - f^*(\mathbf{x}_t, s, t)$ is:

$$\frac{d}{dt} e(s,t) = \frac{d}{dt} f_\theta(\mathbf{x}_t, s, t) - \frac{d}{dt} f^*(\mathbf{x}_t, s, t) = (\partial_t f_\theta + \nabla_{\mathbf{x}_t} f_\theta \cdot \mathbf{u}_t) - 0 = R(t) \tag{21}$$

With boundary condition $e(s,s) = f_\theta(\mathbf{x}_s, s, s) - f^*(\mathbf{x}_s, s, s) = \mathbf{x}_s - \mathbf{x}_s = \mathbf{0}$, integration gives:

$$e(s,t) = \int_s^t R(r)\, dr \tag{22}$$

The integral identity implies $\|e(s,t)\| \le \int_s^t \|R(r)\|\, dr$; it does not require monotonic error growth because residuals may cancel. □

A.5 Equivalence of Corrected Objective (Eq. 8)

We show that replacing the marginal velocity $\mathbf{u}_t$ with the conditional velocity $\mathbf{v}_t$ in the first term of the consistency loss introduces only a parameter-independent constant.

Starting from the context-conditioned consistency objective $\mathcal{L}^u_{\text{consist}} = \mathbb{E}_{\mathbf{x}_t,\mathbf{c}}[\|F_\theta - \mathbf{u}_t + (t-s)\dot{F}_\theta(\mathbf{u}_t)\|^2]$, define the auxiliary objective with $\mathbf{v}_t$ in the first term:

$$\mathcal{L}^v = \mathbb{E}_{\mathbf{x}_t,\mathbf{c},\mathbf{v}_t}\Big[\|F_\theta - \mathbf{v}_t + (t-s)\dot{F}_\theta(\mathbf{u}_\theta)\|^2\Big] \tag{23}$$

Using $\mathbf{v}_t = \mathbf{u}_t + \boldsymbol{\delta}_t$ with $\mathbb{E}[\boldsymbol{\delta}_t \mid \mathbf{x}_t, \mathbf{c}] = \mathbf{0}$:

$$\begin{aligned}\|F_\theta - \mathbf{v}_t + \Delta\|^2 &= \|(F_\theta - \mathbf{u}_t + \Delta) - \boldsymbol{\delta}_t\|^2 \\ &= \|F_\theta - \mathbf{u}_t + \Delta\|^2 - 2(F_\theta - \mathbf{u}_t + \Delta)^\top \boldsymbol{\delta}_t + \|\boldsymbol{\delta}_t\|^2\end{aligned} \tag{24}$$

where $\Delta = (t-s)\dot{F}_\theta(\mathbf{u}_\theta)$. Taking the expectation, the cross term vanishes:

$$\mathbb{E}_{\mathbf{v}_t|\mathbf{x}_t,\mathbf{c}}[(F_\theta - \mathbf{u}_t + \Delta)^\top \boldsymbol{\delta}_t] = (F_\theta - \mathbf{u}_t + \Delta)^\top \underbrace{\mathbb{E}[\boldsymbol{\delta}_t|\mathbf{x}_t, \mathbf{c}]}_{=\mathbf{0}} = 0 \tag{25}$$

Therefore:

$$\mathcal{L}^v = \mathcal{L}^u_{\text{consist}} + \mathbb{E}_{\mathbf{x}_t,\mathbf{c},\mathbf{v}_t}[\|\boldsymbol{\delta}_t\|^2] \tag{26}$$

The added conditional-variance term is independent of θ, so the two regression objectives have the same minimizer. It remains to specify how $\mathbf{u}_t$ in the total derivative term $\dot{F}_\theta(\mathbf{u}_t)$ is estimated. Observe that at $s=t$, the corrected objective (Eq. 8) reduces to the standard FM loss $\|F_\theta(\mathbf{x}_t, t, t) - \mathbf{v}_t\|^2$, whose minimizer is precisely the marginal velocity $\mathbf{u}_t$. The FM component of SnapFlow training (α fraction of each batch) therefore provides a continuously refined estimate $\mathbf{u}_\theta = F_\theta(\mathbf{x}_t, t, t) \approx \mathbf{u}_t$. Substituting $\mathbf{u}_\theta$ for $\mathbf{u}_t$ in the total derivative yields Eq. (8).

B Training and Inference Algorithms

Section 3.4 describes the SnapFlow objective one component at a time; this appendix assembles the components into the concrete procedures that produced the reported checkpoints, so that the training cost and the deployment path can be read off directly. The training loop is given in Algorithm 1 and the inference procedure in Algorithm 2.

Each training step makes three action-expert calls: one student call with gradients and two target calls under stop-gradient that produce the two-step Euler endpoint. The frozen VLM prefix is computed once per batch and its KV cache is reused across all three calls, which is what allows the target to come from the current model itself rather than from an EMA copy or a second teacher network. Memory is dominated by the student call, because only it retains activations for backpropagation; the VLM is frozen, and gradient checkpointing is applied to the trainable action expert.

At deployment the procedure reduces to that of naïve one-step inference: SnapFlow computes the VLM prefix once and evaluates the action expert once. The only difference from the pretrained model is that the destination-time input s is set to 0 rather than to $s=t$, and no target model is retained.

Algorithm 1 SnapFlow Training

Require: Pretrained VLA F_θ, dataset $\mathcal{D}$, ratio α, weight λ, learning rate η, steps N

Initialize destination-time projection $\phi_s \leftarrow \mathbf{0}$; freeze VLM backbone
for $i = 1$ to N **do**
  Sample batch $\{(\mathbf{x}_0^{(j)}, \mathbf{c}^{(j)})\}$ from $\mathcal{D}$
  Sample $\boldsymbol{\epsilon}^{(j)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and source time $t^{(j)}$ from the base-policy schedule
  Compute $\mathbf{x}_t^{(j)} = (1 - t^{(j)})\, \mathbf{x}_0^{(j)} + t^{(j)}\, \boldsymbol{\epsilon}^{(j)}$
  Sample $s^{(j)} = t^{(j)}$ with probability α, otherwise $s^{(j)} = 0$
  $\mathbf{v}^{(1)} \leftarrow \text{sg}(F_\theta(\mathbf{x}_t, t, t \mid \mathbf{c}))$; $\mathbf{x}_{t/2} \leftarrow \mathbf{x}_t - \frac{t}{2}\mathbf{v}^{(1)}$
  $\mathbf{v}^{(2)} \leftarrow \text{sg}(F_\theta(\mathbf{x}_{t/2}, t/2, t/2 \mid \mathbf{c}))$
  $\mathbf{v}_{2\text{E}} \leftarrow \text{clip}((\mathbf{v}^{(1)} + \mathbf{v}^{(2)})/2, -20, 20)$
  $(\mathbf{y}, w) \leftarrow (\boldsymbol{\epsilon} - \mathbf{x}_0, 1)$ if $s = t$, else $(\mathbf{v}_{2\text{E}}, \lambda)$
  $\mathcal{L} \leftarrow w\|F_\theta(\mathbf{x}_t, s, t \mid \mathbf{c}) - \mathbf{y}\|^2$
  Update $\theta \leftarrow \theta - \eta\nabla_\theta\mathcal{L}$ ▷ action expert + ϕ_s only
end for
return Distilled model F_θ

C Per-Task Success Rate Breakdown

The suite averages in Table 1 can hide whether a gain comes from a few tasks or is spread across the benchmark, so this appendix breaks every reported closed-loop number down to

Algorithm 2 SnapFlow 1-NFE Inference

Require: Observation images $\mathbf{o}$, language instruction $\mathbf{l}$, distilled VLA F_θ

1: $\mathbf{c} \leftarrow$ VLM-Prefix$(\mathbf{o}, \mathbf{l})$ ▷ shared VLM computation ($\sim$60 ms)
2: $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^{H \times D}$ ▷ sample noise
3: $\hat{\mathbf{x}}_0 = \mathbf{x}_1 - F_\theta(\mathbf{x}_1, s{=}0, t{=}1 \mid \mathbf{c})$ ▷ one action-expert evaluation ($\sim$24 ms)
4: Execute first n_{act} steps of $\hat{\mathbf{x}}_0$

the task level; the aggregate values in the main text can be recomputed from these tables. The per-task outcomes are listed in Table 5 for $\pi0.5$ and Table 6 for $\pi0$. The archived $\pi0.5$ table reports 10 episodes per task, and the separate $\pi0$ table uses matched initial-state identifiers for its paired comparisons; task-level percentages correspond to the episodes in the respective evaluation, and the two sets of results should not be pooled.

On archived $\pi0.5$ results, SnapFlow is at least as high as the teacher on all 40 task averages: it improves three tasks and matches the teacher on the remaining 37, with a largest increase of 20 points (libero_goal Task 3) and no decreases. On $\pi0$, SnapFlow improves on 22 of 40 tasks with gains of up to 60 points (libero_10 Task 9) and regresses on six; the eight tasks where the teacher scores below 50% improve by 21 points on average.

Table 5: **Per-task LIBERO success rate (%) for $\pi0.5$.** Archived aggregate evaluation with 10 episodes per task; Base is the 10-NFE teacher, SF is one-NFE SnapFlow.

	libero_spatial			**libero_object**			**libero_goal**			**libero_10** (long)		
Task	Base	Naïve	SF	Base	Naïve	SF	Base	Naïve	SF	Base	Naïve	SF
0	100	100	100	100	100	100	100	100	100	90	90	90
1	100	100	100	100	100	100	100	100	100	100	100	100
2	100	100	100	100	100	100	90	90	90	100	100	100
3	100	100	100	100	90	100	80	90	100	100	100	100
4	100	100	100	100	100	100	100	100	100	100	100	100
5	90	70	90	100	100	100	100	100	100	100	100	100
6	100	100	100	100	100	100	100	100	100	100	90	100
7	90	100	100	100	100	100	100	100	100	100	100	100
8	100	100	100	100	100	100	100	100	100	80	80	80
9	100	90	100	100	100	100	90	100	100	100	80	100
Avg	98.0	96.0	99.0	100.0	99.0	100.0	96.0	98.0	99.0	97.0	94.0	97.0

Table 6: **Per-task LIBERO success rate (%) for $\pi0$.** Separate paired evaluation under the training and evaluation configuration used here; its protocol is distinct from the archived $\pi0.5$ results.

	libero_spatial			**libero_object**			**libero_goal**			**libero_10** (long)		
Task	Base	Naïve	SF	Base	Naïve	SF	Base	Naïve	SF	Base	Naïve	SF
0	80	90	90	70	100	100	90	60	90	50	40	50
1	70	100	100	60	100	80	90	100	100	40	80	70
2	100	100	100	90	100	100	100	90	70	80	80	80
3	90	100	100	90	50	80	80	90	80	70	90	90
4	100	100	100	100	100	100	70	80	100	30	10	40
5	10	10	20	60	30	60	80	90	80	70	70	100
6	90	100	100	100	100	90	60	100	80	40	40	60
7	70	100	100	90	70	80	100	100	100	10	10	0
8	60	80	80	100	100	100	80	100	100	30	20	30
9	40	90	90	90	100	80	70	80	90	20	60	80
Avg	71.0	87.0	88.0	85.0	85.0	87.0	82.0	89.0	89.0	44.0	50.0	60.0

D LIBERO Success Rate Visualization

Table 1 presents the closed-loop results numerically; to make the suite-level pattern easier to compare across the three methods and the two models, this appendix plots the same numbers, as shown in Figure 4. The left panel makes visible how close all three methods are on the near-ceiling $\pi 0.5$ teacher, whereas the right panel shows that both one-step variants move well above the $\pi 0$ teacher, with SnapFlow matching or exceeding naïve one-step on every suite.

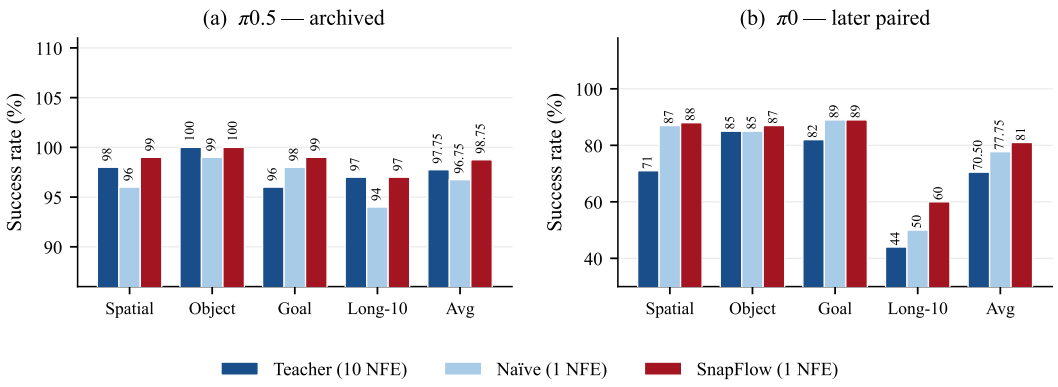

Figure 4: **LIBERO success rate by suite.** The left panel reports the archived $\pi 0.5$ evaluation; the right panel reports the separate $\pi 0$ paired evaluation under the configuration used here. Bars show the 10-NFE teacher, naïve one-step inference, and one-NFE SnapFlow.

E Detailed Offline Metrics Analysis

A lower mean error could in principle come from improving easy samples while leaving hard ones unchanged, which would matter for a robot policy because rare large errors are the ones that cause failures. This appendix therefore reads Table 2 of the main text beyond the mean, looking at the median, the spread, and the upper percentiles of the per-sample error.

On $\pi 0.5$, SnapFlow reduces mean MSE by 33.9%, median MSE by 7.6%, P90 by 23.6%, and P95 by 29.4%, while the standard deviation decreases by 45.2% and CosSim increases from .9885 to .9916. The median moves least and the upper percentiles most, so the improvement is concentrated in the tail rather than in typical samples. The SmolVLA offline evaluation shows the same pattern at a smaller scale, with lower mean, P90, and standard deviation under one-NFE SnapFlow (MSE -8.3%, P90 -11.4%, std. -12.6%), which indicates that the effect is not specific to the PaliGemma-based architecture. Offline error and closed-loop success measure different properties, but the archived $\pi 0.5$ result improves both the aggregate success rate and the held-out error statistics.

F Positioning Against Concurrent VLA Acceleration Methods

Several concurrent works also target VLA inference efficiency, but they compress different parts of the computation graph. Table 7 summarizes the mechanisms. The reported success changes and speedups are taken from the respective papers, which use different base models and evaluation settings, so the table positions the mechanisms rather than benchmarking them head to head.

Shallow-π and EfficientVLA reduce the cost of each transformer pass, whereas SnapFlow removes nine of the ten passes. The two axes are in principle composable, although we have not evaluated their combination.

Table 7: **Mechanisms of concurrent VLA acceleration methods.** Numbers follow the cited papers and are not measured under a common protocol.

Method	Layers (architecture)	Steps (sampling)	Success Δ	Model speedup	Mechanism
Shallow-π (Jeon et al., 2026)	$18 \to 6$	10	< 1 pp drop	$> 2\times$	Layer distillation
EfficientVLA (Yang et al., 2025)	$28 \to 22$	unchanged	-0.6 pp	$1.93\times$	Layer/token pruning + caching
SnapFlow	unchanged	$10 \to 1$	$+1.0$ pp (archived $\pi 0.5$)	$3.3\times$	Destination-time distillation

G Latency Decomposition Details

The main text reports a single speedup number per model, but understanding where the remaining latency goes requires separating the two components of a forward pass: the VLM prefix, computed once per chunk, and the action expert, evaluated once per step. This appendix measures both components at every step count so that the speedup can be attributed, as shown in Figure 5 and, with exact numbers, in Table 8.

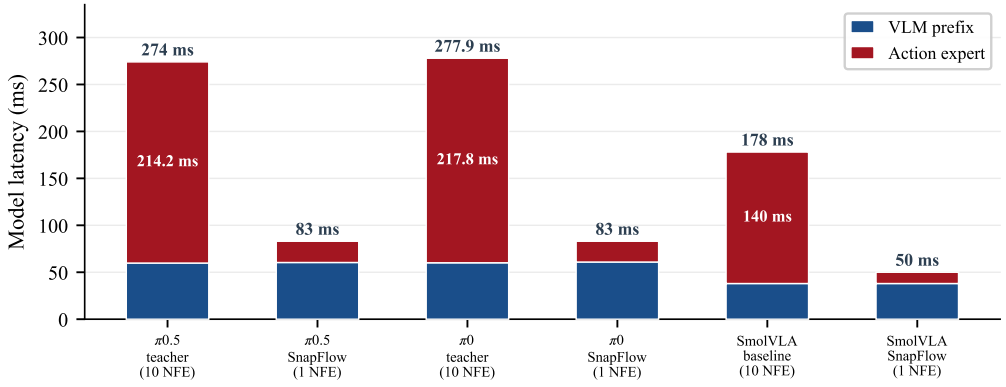

Figure 5: **Latency decomposition: VLM prefix and action expert.** SnapFlow reduces the action expert from 10 sequential evaluations to one, which leaves the fixed VLM prefix as the dominant cost. Batch-one model speedup is about $3.3\times$ for $\pi 0.5$ and $3.35\times$ for $\pi 0$.

All measurements are synchronized batch-one inference; the archived $\pi 0.5$ numbers were taken on one NVIDIA A800-80G, and the separate $\pi 0$ numbers use the corresponding synchronized timing protocol. On $\pi 0.5$, the 10-step action expert accounts for roughly 78% of model latency (about 214 ms out of 274 ms), at approximately 23 ms per step, while the VLM prefix costs about 60 ms regardless of the step count. Reducing the step count therefore lowers the sequential action-generation cost from $O(K)$ evaluations to one, and at one NFE the fixed VLM prefix becomes the dominant component; this constant overhead is what bounds the measured model-path speedup at about $3.3\times$ rather than the $10\times$ reduction in expert evaluations.

H Multi-Step Inference with the Distilled Model

The FM branch of the SnapFlow objective (Eq. 12) keeps training the local velocity field at $s = t$, so the distilled network can still be integrated with several Euler steps. Table 9 evaluates the archived $\pi 0.5$ SnapFlow checkpoint at 1, 2, 3, 5, and 10 action-expert evaluations and compares it with the pretrained teacher integrated with the same step counts. All ten rows use the same 500 held-out LIBERO samples and the same noise draws, so comparisons within the table are paired; the sample draw differs from Table 2, which is why the one-NFE and ten-NFE values do not coincide exactly with that table.

Two observations follow. First, the distilled model has lower mean and P90 error than the teacher at every step count, including ten steps, so the shortcut branch does not degrade the local velocity field that the FM branch maintains. Second, its error is lowest at two

Table 8: **Model inference latency vs. NFE** (A800, batch size 1). The baseline rows sweep the pretrained solver; the SnapFlow row is the distilled one-NFE model. Speedups are relative to the same model at 10 NFE.

VLA	NFE	Model (ms)	Action Fraction	Model Speedup
π0.5 baseline	1	81.0	26%	3.38×
	2	103.3	42%	2.65×
	3	124.4	52%	2.20×
	5	166.9	64%	1.64×
	10	274.0	78%	1.00×
π0.5 SnapFlow	1	83	27%	3.3×
π0 baseline	1	82.8	28%	3.36×
	2	104.7	44%	2.65×
	3	126.3	54%	2.20×
	5	169.3	66%	1.64×
	10	277.9	80%	1.00×
π0 SnapFlow	1	83.0	29%	3.35×
SmolVLA (0.5B)	1	50	24%	3.56×
	10	178	79%	1.00×

Table 9: **Offline error versus number of Euler steps** (π0.5, 500 held-out samples, shared noise). Teacher rows truncate the pretrained solver; SnapFlow rows integrate the distilled model with the same step counts, using $s = 0$ only for the one-NFE row.

	Teacher (pretrained solver)			**SnapFlow (distilled)**		
NFE	MSE↓	P90 MSE↓	CosSim↑	MSE↓	P90 MSE↓	CosSim↑
1	.00892	.01424	.9911	.00826	.01296	.9911
2	.00918	.01368	.9910	.00723	.01064	.9920
3	.00957	.01342	.9906	**.00716**	**.00989**	**.9920**
5	.01116	.01425	.9888	.00772	.01045	.9917
10	.01238	.01672	.9879	.00864	.01324	.9910

to three steps and rises slowly thereafter, mirroring the monotone increase of the teacher's error with step count that Theorem 3 anticipates; the additional steps buy no accuracy on this held-out set. One-NFE inference therefore sits within .001 of the distilled model's best operating point while removing all but one action-expert evaluation.

I Action Execution Horizon

Closed-loop success depends not only on the quality of a chunk but on how much of it is executed before the next inference. Since one-NFE inference is 3.3× cheaper per call, it could in principle be spent on more frequent replanning rather than on a lower control latency. We test this by sweeping the executed horizon $n_{\text{act}} \in \{1, 3, 5, 10, 20\}$ on libero_10, holding the archived evaluation setup fixed and comparing the 10-NFE teacher with one-NFE SnapFlow (Figure 6).

The sweep trades the lead across horizons. The archived sweep gives teacher success of 97%, 97%, 90%, 88%, 77% and SnapFlow success of 92%, 97%, 93%, 87%, 72% at $n_{\text{act}} = 20, 10, 5, 3, 1$, respectively; the $n_{\text{act}} = 10$ values coincide with the libero_10 column of Table 1. SnapFlow is ahead at horizon 5 and level with the teacher at horizon 10 in this

archived sweep; the separate $\pi 0$ evaluation does not include an n_{act} sweep, so these results do not support a monotonic horizon claim.

The largest archived gap is at $n_{\text{act}} = 20$, where the teacher leads by five points; at $n_{\text{act}} = 5$ SnapFlow leads by three points. These differences are task- and protocol-dependent, so the sweep is best read as a sensitivity analysis rather than as evidence for a universal optimal horizon.

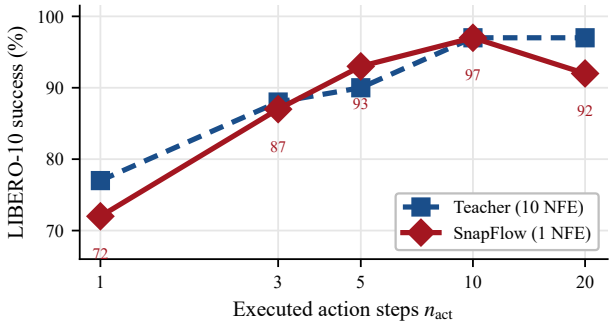

Figure 6: **Sensitivity to the executed action horizon** (archived $\pi 0.5$ libero_10 sweep). The two methods trade the lead across horizons, with SnapFlow ahead at 5 executed steps and level with the teacher at 10.

J Training Dynamics

Because the shortcut target is generated by the model being trained, a natural concern is whether the self-referential objective is stable early in training, before the destination-time projection has moved away from zero. The archived $\pi 0.5$ convergence study addresses this over the first 5k updates (batch size 4, one A800, approximately 25 min): the gradient norm peaks at 7.48 at step 650 and remains finite thereafter, with no divergence. No raw curve is included because the archived release retained only these aggregate convergence statistics.

K Training Hyperparameters

To make the reported checkpoints reproducible, this appendix records the complete training and inference configuration, as listed in Table 10 for the archived $\pi 0.5$ run. The separate $\pi 0$ run uses the same optimizer, coefficients ($\alpha = 0.5$, $\lambda = 0.1$), and clip range, with 15k updates; its reported results are specific to this training and evaluation configuration. The offline ablation in Table 3 varies one choice at a time around the archived default.

L Real-Robot Deployment Details

Section 4.5 shows a single shelf configuration and states the conclusions we draw from the real-robot deployment; this appendix records the platform, task definition, and training procedure behind that deployment so that its scope is clear, and shows the shelf configurations that do not appear in the main text.

The robot is a dual-arm AgiBot humanoid with parallel-jaw grippers, operating in front of pharmacy-style retail shelving stocked with boxed and bottled products. Each shelf configuration has one designated target product; an episode starts with the arms at rest and is complete when the item has been grasped and pulled clear of the shelf. The configurations differ in shelf height, product density, and the surrounding items, so the policy has to reach into a cluttered opening without disturbing neighbouring packages.

For each shelf configuration we train a separate teacher–SnapFlow pair from the same $\pi 0.5$ checkpoint. The teacher is fine-tuned on teleoperated demonstrations of retrieving that



Figure 7: **Additional real-robot shelf configurations.** Keyframes at matched fractions of each episode for the 10-NFE teacher and one-NFE SnapFlow on three further shelf layouts (labelled B, C, and D; layout A appears in Figure 3). In every pair both policies reach into the shelf, grasp the target product, and extract it. The recordings are a deployment demonstration and are not used for quantitative claims.

configuration's target with the standard flow-matching objective and deployed with its default 10-step Euler integration. SnapFlow is initialized from this fine-tuned teacher and distilled with the objective of Section 3.4, using the same $\alpha = 0.5$ and $\lambda = 0.1$ as the simulation experiments, the zero-initialized destination-time projection, a frozen VLM backbone, and

Table 10: **SnapFlow training hyperparameters.**

Parameter	Value
SnapFlow	
FM/Consistency ratio α	0.5
Consistency weight λ	0.1
Shortcut-target clamp range	$[-20, 20]$
Target-time projection	Zero-init linear layer
Training	
Optimizer	AdamW
Learning rate	2.5×10^{-5}
Gradient clipping norm	1.0
Warmup steps	500
Total steps	30,000 (π0.5) / 15,000 (π0)
Batch size	4
Precision	bfloat16
Frozen components	VLM backbone (PaliGemma)
Trainable components	Action expert (about 300M) + target-time proj
Gradient checkpointing	Enabled
Archived π0.5 wall time	approximately 12 h for 30,000 updates
Inference	
Denoising steps	1 (1-NFE)
Action chunk size	50
Executed action steps	10

a trainable action expert; it is deployed with one action-expert evaluation ($s = 0$, $t = 1$). Neither policy uses an external teacher network or critic at deployment.

The recordings cover nine episodes across four shelf configurations: SnapFlow on all four and the teacher on all four, with a second SnapFlow episode on one configuration. They were made once fine-tuning had converged on each target, and neither policy failed to grasp its product in these sessions; however, the sessions were not organized as a paired protocol with fixed initial states and pre-registered trial counts, so we present the material as a qualitative deployment check and make no success-rate, completion-time, or latency claims from it. The three shelf configurations not included in Figure 3 are shown in Figure 7, again sampled at matched fractions of each episode, and the full recordings are indexed in Appendix M.

M Supplementary Video Index

Keyframes can only suggest how a policy moves, so the supplementary archive provides the complete videos behind the real-robot figures together with side-by-side simulation rollouts that make the speed difference between the two policies visible. All clips are unedited apart from trimming to the manipulation segment; no frames are removed or resampled within an episode.

The folder `real_robot/` contains the nine real-robot recordings, each a 1280×720 clip at 60 fps from a fixed third-person camera. Files are named by shelf label and policy, for example `shelfA_snapflow.mp4` and `shelfA_teacher.mp4`; shelf A appears in Figure 3, shelves B–D in Figure 7, and `shelfB_snapflow_run2.mp4` is the second SnapFlow episode on shelf B.

The folder `libero_side_by_side/` contains thirty clips from the archived π0.5 LIBERO evaluation, rendered at 30 fps. Each clip pairs an episode in which both policies succeed, with the 10-NFE π0.5 teacher on the left and one-NFE SnapFlow on the right, started from the same initial state and sharing an elapsed-time counter so that the two rollouts can be compared frame by frame. The per-chunk latencies printed in the header (274 ms and 83 ms) are the archived A800 batch-one measurements from Table 8; because the simulator does not run in real time, the elapsed-time counter is derived from policy steps and this per-chunk latency rather than from wall-clock time.