RAP: RECURSIVE ACTION-PREVIEW FLOW MATCHING FOR FAST AND ROBUST VISUOMOTOR CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent diffusion and flow-matching methods enable few-step action generation by initializing from temporally grounded sources derived from historical actions or previous action forecasts. However, source construction can differ between offline training and deployment: training sources rely on expert or ground-truth actions, whereas deployment sources come from the policy's own executed or predicted actions. This train–deployment source mismatch can make the learned source-to-action mapping unreliable as deployment sources deviate from the training source distribution, degrading policy performance and robustness. Adding Gaussian noise to temporal sources can improve robustness, but requires manual tuning of the noise scale. We propose Recursive Action-Preview Flow Matching (RAP) to align source construction between training and deployment. RAP carries forward the unexecuted part of the previous action forecast as an action preview to construct the source for the next action-generation cycle, while the latest observation guides its refinement. During training, RAP recursively constructs sources from detached policy-generated action previews, matching the source formation used at deployment without additional Gaussian noise injection. Across 8 simulated tasks from 4 benchmarks and 2 real-world tasks, RAP achieves strong performance with only 1 or 3 inference steps. Under large initial-pose perturbations, RAP improves success by 22–50 percentage points over noise-free baselines while remaining competitive with methods using Gaussian source perturbations. RAP combines fast action generation with robustness to both initial-pose changes and visual uncertainty.

1 INTRODUCTION

Visuomotor policies map visual observations and robot states to control actions, often by predicting an *action chunk* that covers several future control steps (Zhao et al., 2023; Black et al., 2025). Diffusion and flow-matching policies typically generate these action chunks from Gaussian noise through iterative denoising or numerical integration (Chi et al., 2025; Lipman et al., 2023). Each denoising or integration step requires a function evaluation, so reducing the number of function evaluations (NFE) is important for efficient action generation.

Recent work pursues this goal in two main ways. One line of work uses consistency learning or distillation to reduce the number of denoising or integration steps while retaining Gaussian initialization (Prasad et al., 2024; Gao et al., 2024; Li et al., 2026b). Another line of work redesigns the source distribution to provide a more informative starting point for action generation (Chen et al., 2024; Gao et al., 2026; Jia et al., 2026). Among these source-based approaches, *temporal-source policies* initialize action generation from temporally grounded sources, such as recent action histories or previous action forecasts (Jia et al., 2026; Kang et al., 2026). By carrying temporal information from previous actions or predictions, these sources shorten the generation path to the next action chunk and enable effective generation with fewer inference steps.

However, temporal-source policies can suffer from a train–deployment mismatch in source construction. During offline training, sources are constructed from expert action sequences or ground-truth future actions, whereas at deployment they are generated from the policy's own executed or predicted actions. As deployment sources deviate from the training source distribution, the learned source-to-action mapping can become unreliable, degrading policy performance and robustness. One way to improve robustness is to add Gaussian perturbations to training sources to broaden their coverage

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

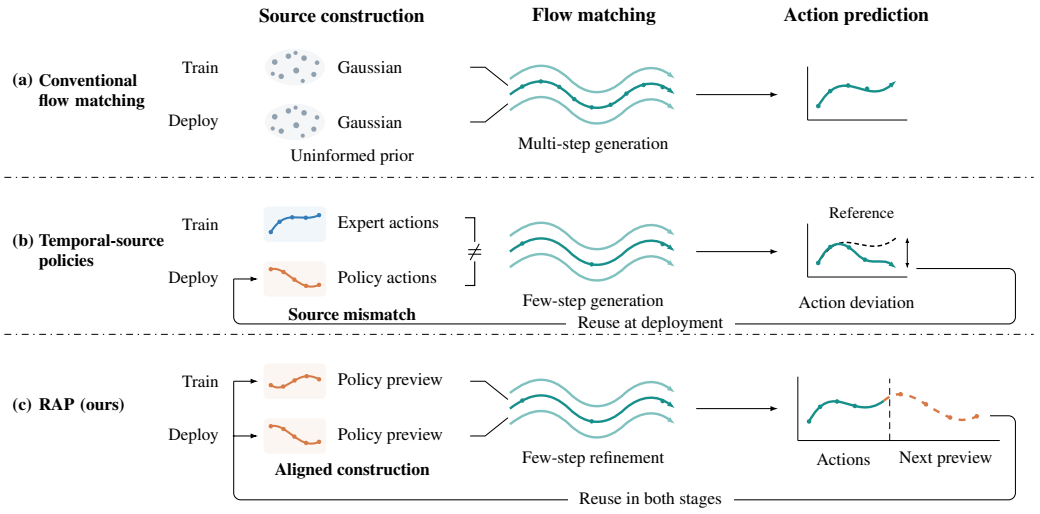


Figure 1: Source construction in generative visuomotor policies. (a) Conventional flow matching starts from an uninformed Gaussian prior in both training and deployment. (b) Temporal-source policies (A2A, WarmPrior) enable few-step generation, but their sources change from expert actions during training to policy-generated actions or forecasts at deployment. This mismatch can make the learned source-to-action mapping unreliable. (c) RAP carries forward the unexecuted part of each forecast as an action preview in both stages. Aligning source construction during training with that used at deployment lets the policy learn to refine its own previews using the latest observation.

(Jia et al., 2026; Kang et al., 2026). However, this approach has two limitations. The noise scale must be manually tuned, and independently sampled Gaussian perturbations do not necessarily match the distribution of policy-generated source errors, which depend on the current state and previous predictions and can exhibit temporal correlation.

These limitations suggest a more direct solution: align source formation during offline training with the process used at deployment. A practical challenge is that policy-generated temporal sources are not directly available in offline demonstrations. To address this challenge, our key insight is that training sources can be generated recursively by applying the current policy to consecutive observations from the demonstrations. Based on this insight, we propose *Recursive Action-Preview Flow Matching* (RAP). In action generation, RAP retains the unexecuted part of its action forecast as an *action preview*. This preview is carried forward to construct the source for the next action-generation cycle, where the policy conditions on the latest observation to refine the preview and generate the next action chunk.

RAP incorporates two key designs. First, we introduce *Self-Generated Recursive Source Training* (SG-RST). Given consecutive observation windows from an expert trajectory, RAP generates an action preview from the earlier window, detaches it, and uses it to construct the source for the following window, while the corresponding expert action remains the supervision target. Repeating this process along the trajectory reproduces the recursive source formation used at deployment without additional environment interaction. Second, recursive preview reuse requires visual information from two different times: the preview is generated under an earlier observation but must be refined according to the latest one. We therefore introduce *Temporal Visual Fusion* (TVF), which fuses the cached visual feature from preview generation with the current visual feature through a learned linear projection. This provides both the context in which the preview was generated and the latest state feedback for refining it, while avoiding an additional visual-encoder pass.

We evaluate RAP on 8 simulated tasks across 4 benchmarks and 2 real-world manipulation tasks using a Franka arm. RAP policies using only 1 or 3 inference steps achieve the best or tied-best result on each simulated task and the highest success rates on both real-world tasks. RAP improves robustness to initial pose uncertainty while retaining visual robustness comparable to A2A. Under large initial-pose perturbations, it improves success by 22–50 percentage points over noise-free

baselines and remains competitive with methods using Gaussian source perturbations. Source-consistency analysis shows a 64% smaller mean paired source gap than WP-Preview and temporal correlations in deployment preview errors that independent Gaussian perturbations do not capture.

Our main contributions are summarized as follows:

Train–deployment source mismatch. We identify a train–deployment mismatch in temporally grounded sources, where expert-derived sources used during offline training differ systematically from policy-generated sources at deployment. We further show that deployment preview errors exhibit temporal correlations that are not captured by independent Gaussian perturbations. These observations motivate aligning training source formation directly with the process used at deployment.

Recursive Action-Preview Flow Matching. We propose RAP, which recursively constructs training sources from detached policy-generated action previews to match the source formation used at deployment. SG-RST enables this alignment using only offline demonstrations, avoiding the need for additional Gaussian perturbations in source construction. TVF further combines the visual context of preview generation with the latest state feedback to refine the action preview.

Fast and robust visuomotor control. Experiments on 8 simulated tasks across 4 benchmarks and 2 real-world tasks demonstrate strong performance with few inference steps. RAP improves robustness to initial pose uncertainty while retaining visual robustness. Under large initial-pose perturbations, it improves success by 22–50 percentage points over noise-free baselines and remains competitive with methods using Gaussian source perturbations.

2 Related Work

Few-step generative visuomotor policies. Few-step generative policies reduce inference cost in two different ways. Consistency- and distillation-based methods compress the transformation from a standard Gaussian source to the action distribution into fewer denoising or integration steps (Prasad et al., 2024; Gao et al., 2024; Zhang et al., 2025; Li et al., 2026b). This lowers NFE, but the policy still starts from an uninformed random source. Instead, a second strategy makes the source itself more informative. BRIDGER starts from the output of a source policy and refines it, VITA uses visual representations as the flow source for latent action generation, and LeaP learns a proprioception-conditioned Gaussian source (Chen et al., 2024; Gao et al., 2026; Dai et al., 2026). By incorporating task-relevant information into the source, these methods reduce the burden on the subsequent generative process. RAP follows this second direction and further exploits temporal information carried across consecutive control cycles.

Temporal-source policies. Temporal-source policies exploit information from previous control cycles to initialize subsequent action generation. A2A uses recent action history, Temporal Policy uses robot-state history, and Streaming Flow Policy samples around previous actions while producing actions during flow integration (Jia et al., 2026; Miller & Jagersand, 2026; Jiang et al., 2025). STEP predicts a warm-start sequence from the current observation and the previous action sequence (Li et al., 2026a). WarmPrior constructs temporal sources from either the preceding action chunk (WP-Past) or a previous action forecast (WP-Preview), together with residual Gaussian perturbations (Kang et al., 2026). These methods show that temporal information can provide an informative initialization for few-step action generation. However, when temporal sources depend on previous policy outputs or executed actions, their construction at deployment can differ from that available in offline demonstrations. Instead, RAP recursively generates action previews with the current policy during offline training and uses detached previews to construct subsequent sources, aligning source formation during offline training with the process used at deployment.

3 Preliminaries

Conditional flow matching (Lipman et al., 2023) learns to transport samples from a source distribution $p_0(\cdot \mid c)$ to a target distribution $p_1(\cdot \mid c)$ under condition c. A vector field v_θ defines the transport via

$$\frac{\mathrm{d}z_t}{\mathrm{d}t} = v_\theta(z_t, t, c), \qquad t \in [0, 1], \tag{1}$$

where $z_0 \sim p_0(\cdot \mid c)$ is the source sample and t is the continuous flow time.

Training. For a paired source, target, and condition (z_0, z_1, c) with $z_1 \sim p_1(\cdot \mid c)$, we sample $t \sim \mathcal{U}(0,1)$ and interpolate along the linear path $z_t = (1-t)z_0 + tz_1$. The velocity along this path is $u_t = z_1 - z_0$. The flow-matching objective trains the vector field to predict this velocity:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}\Big[\|v_\theta(z_t, t, c) - u_t\|_2^2\Big]. \tag{2}$$

Inference. Given a source and a condition, action generation follows the learned vector field from flow time 0 to 1. Starting from $z^{(0)} = z_0$, an explicit Euler solver with K steps applies

$$z^{(j+1)} = z^{(j)} + \Delta t\, v_\theta(z^{(j)}, t^{(j)}, c), \qquad t^{(j)} = j/K, \quad \Delta t = 1/K, \tag{3}$$

for $j = 0, \ldots, K-1$. Each step evaluates the vector field once, so the $\mathrm{NFE} = K$.

4 RAP: Recursive Action-Preview Flow Matching

RAP addresses the challenge of aligning training and deployment source distributions when deployment sources depend on the policy's own earlier predictions. Offline demonstrations provide expert actions, but do not directly provide these policy-generated sources. Our approach is to reproduce recursive preview reuse during offline training and supervise the predictions with expert actions. We first describe how previews connect successive action predictions (Section 4.1), then introduce SG-RST, which combines recursive preview generation with expert supervision (Section 4.2). Finally, we describe the visual conditioning and learning objectives used to refine these previews (Section 4.3).

4.1 Recursive Action Previews

At each decision cycle k, RAP uses the cached action preview P_k and the latest observation to predict $2H$ actions. The first H actions, $\hat{A}_k$, are executed; the remaining H, $\hat{P}_{k+1}$, are cached as the preview for the next cycle. Figure 2 shows how this preview aligns with the next execution interval.

The cached preview covers the current H-step interval but provides no actions for the following one. We repeat its final action for another H steps to form S_k. This padding supplies a constant-action continuation from the preview's endpoint, aligning the two source segments with the current-action and next-preview segments of the expert target Y_k. The policy uses the full source and visual context to refine the current actions and predict the next preview. At episode start, we initialize S_k by repeating the action corresponding to the current robot state. No Gaussian perturbation is added.

Following A2A (Jia et al., 2026), we perform flow matching in the action latent space. The same action encoder E_a maps the source and expert target into a shared 512-dimensional space:

$$z_0 = E_a(S_k), \qquad z_1 = E_a(Y_k). \tag{4}$$

Conditioned on visual features, K Euler steps transform z_0 into the predicted endpoint $\hat{z}_1$, which the action decoder D_a converts into $\hat{A}_k$ and $\hat{P}_{k+1}$ (Figure 3). These modules are trained jointly; the expert target is used only during training. Appendix A gives the implementation details.

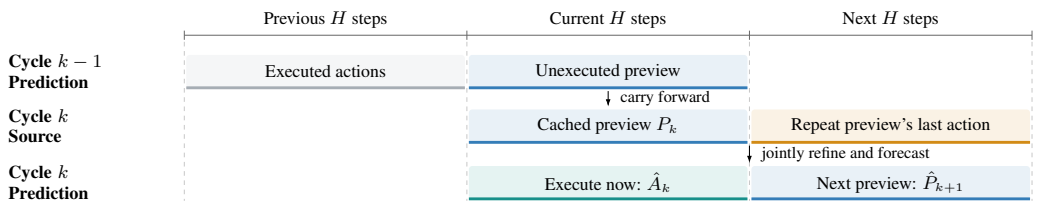


Figure 2: **Temporal alignment of recursive action previews.** The unexecuted half of the previous prediction covers the current H-step interval. Repeating its final action initializes the following interval with a constant-action continuation. The resulting source spans the same two intervals as the target: RAP jointly refines the current actions and predicts the next preview.

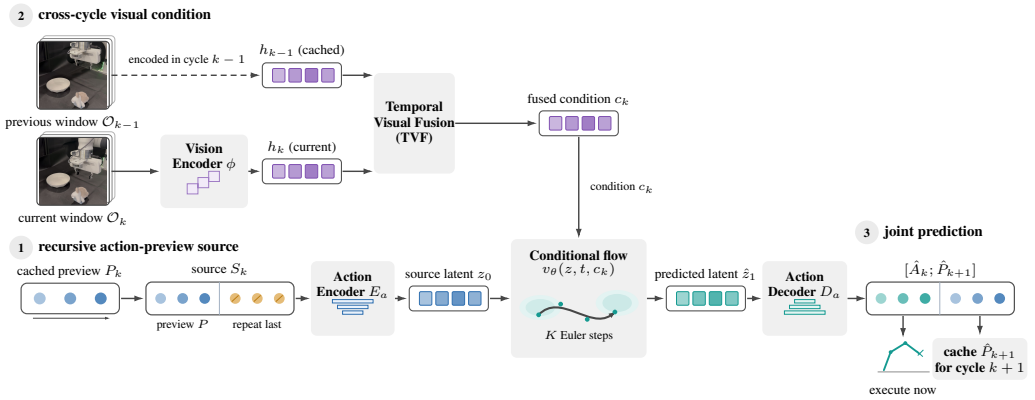


Figure 3: **Online prediction in RAP.** The cached preview is extended into a source, encoded by E_a, refined through K latent flow steps, and decoded by D_a. TVF combines cached and current visual features to condition the prediction. The first action chunk is executed, while the next preview and current visual feature are cached for the following cycle. During training, the same E_a also encodes the expert target.

4.2 Self-Generated Recursive Source Training

Each policy-generated preview depends on earlier predictions. Reconstructing their full history for every training sample would require replaying the current policy from the episode start, with increasing cost for later samples. *Self-Generated Recursive Source Training* (SG-RST) approximates this recursion using at most $J = 2$ preceding policy calls.

Given three consecutive demonstration windows, we initialize the source from the robot state in the first window and make two predictions, passing each preview and visual feature to the next call (Figure 4). Thus, the supervised call receives a preview produced through recursive reuse. The preceding predictions and visual features are detached. For the first action chunk, we directly use the state-based initializer and the initial visual feature for both TVF inputs. The second chunk requires only one preceding call; later chunks use two.

The final call predicts the current actions and next preview, both supervised by corresponding expert actions through the RAP objective (Section 4.3). Only this call receives gradients and updates the shared policy. Subsequent samples regenerate their sources with the updated policy, exposing training to its changing prediction errors. By passing policy-generated previews between recorded observation windows without executing predicted actions, SG-RST approximates deployment recursion entirely offline. Appendix C gives the pseudocode.

4.3 RAP Policy Learning

Temporal visual fusion. An action preview is generated under an earlier observation and must be refined using the latest one. Supplying both visual contexts allows the policy to account for changes in the scene when updating its forecast. *Temporal Visual Fusion* (TVF) combines the cached feature from preview generation with the current visual feature through a learned linear projection:

$$c_k = W_c[h_{k-1}; h_k], \qquad h_k = \phi(\mathcal{O}_k), \tag{5}$$

where $\mathcal{O}_k$ is the current observation window, ϕ is the visual encoder, and $[\cdot;\cdot]$ denotes concatenation. Since h_{k-1} was already computed when the preview was generated, it can be reused from the cache (Figures 3 and 4). TVF therefore uses features from two observation windows with only one new visual-encoder pass per decision, avoiding the cost of re-encoding the earlier window. The current feature h_k is then cached for the next cycle; at initialization, both TVF inputs use the current feature.

Learning objective. SG-RST uses A2A's learning objective (Jia et al., 2026). The flow-matching loss $\mathcal{L}_{\text{FM}}$ supervises the latent flow, and the autoencoder loss $\mathcal{L}_{\text{AE}}$ reconstructs expert actions. The

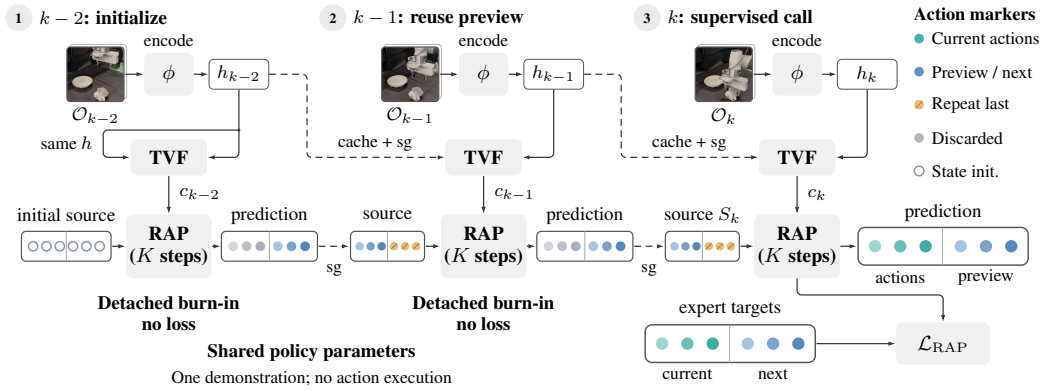


Figure 4: **Self-Generated Recursive Source Training (SG-RST).** SG-RST trains RAP to refine its own previews using offline demonstrations. In the illustrated $J = 2$ case, two recursive policy calls starting from a state-based source produce the preview and cached visual feature for the supervised call. This call jointly predicts current actions and the next preview, using expert actions as targets for the learning objective $\mathcal{L}_{\text{RAP}}$. All calls share the same policy parameters, which are updated through the supervised call while gradients through the preceding calls are stopped (sg).

inference-consistency loss $\mathcal{L}_{\text{IC}}$ supervises the K-step prediction in latent and action space, helping prevent latent action space collapse (Gao et al., 2026). The total objective is

$$\mathcal{L}_{\text{RAP}} = \lambda_{\text{FM}}\mathcal{L}_{\text{FM}} + \lambda_{\text{AE}}\mathcal{L}_{\text{AE}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}}. \tag{6}$$

Training uses K Euler steps for both recursive and supervised predictions; deployment uses the same K. Loss definitions and weights are in Appendix A.2, and inference is summarized in Algorithm 2.

5 Experiments

We evaluate RAP on eight simulation tasks from RLBench (James et al., 2020), ManiSkill (Mu et al., 2021), LIBERO (Liu et al., 2023), and robomimic (Mandlekar et al., 2022), and on Pick-Place Doll and Press Bell using a Franka arm with third-person and wrist-mounted cameras. Actions are joint angles in simulation and end-effector states on the real robot. We compare with A2A (Jia et al., 2026) and WP-Preview (Kang et al., 2026), which initialize generation from action history and previews, respectively; VITA (Gao et al., 2026), which uses a visual latent; and FM-UNet (Lipman et al., 2023) and DDIM-UNet (Chi et al., 2025; Song et al., 2021), which start from Gaussian noise. Table 1 reports task success in both settings at each method's inference budget. Figure 5 illustrates the tasks. Implementation and evaluation settings are in Appendix A.

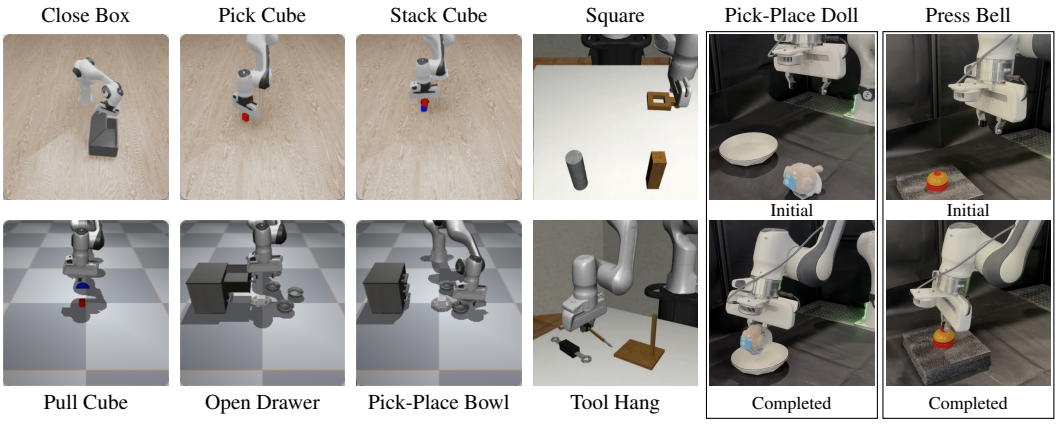


Figure 5: Simulation and real-robot tasks. The first four columns show the eight simulation tasks; the last two show initial (top) and completed (bottom) states for Pick-Place Doll and Press Bell.

Table 1: Task success rate (%) in simulation and on the real robot. Simulation uses 100 rollouts per cell after 30 training epochs, except Tool Hang (†: 200 epochs). Each real-robot task uses 50 demonstrations, 200 training epochs, and 20 trials per method. A dash denotes an unevaluated configuration. Bold and underline mark the best and second-best distinct values in each row.

Task	Method	RAP		WP-Preview		A2A	VITA	FM-UNet	DDIM-UNet
	NFE	1	3	1	3	6	6	10	40
Simulation									
Close Box		89	**97**	89	89	<u>92</u>	88	82	70
Pick Cube		**100**	<u>97</u>	<u>97</u>	**100**	92	88	70	56
Stack Cube		**97**	94	83	<u>95</u>	86	80	28	36
Pull Cube		90	**93**	88	**93**	<u>92</u>	66	76	76
Open Drawer		**92**	88	84	88	**92**	<u>90</u>	34	64
Pick-Place Bowl		74	**86**	79	78	<u>82</u>	80	56	60
Square		<u>79</u>	**84**	70	37	67	15	37	23
Tool Hang†		**31**	20	21	18	<u>23</u>	18	7	20
Real robot									
Pick-Place Doll		–	**70**	–	<u>65</u>	55	–	–	45
Press Bell		–	**75**	–	<u>65</u>	<u>65</u>	–	–	60

At least one RAP configuration achieves the best or tied-best result on every simulated task, and three-NFE RAP has the highest observed success on both real-robot tasks. These results show that preview refinement supports strong task performance with only one or three function evaluations. The advantage over WP-Preview on both physical tasks is particularly relevant: the two methods reuse predicted previews at deployment and use the same three-NFE budget, while RAP learns to refine policy-generated sources during training. The following experiments examine robustness and source consistency. Appendix A.5 discusses variation across inference budgets, and Appendix A.4 reports measured inference latency.

5.1 Robustness under Uncertainty

Action-to-action policies reuse information from earlier predictions or executed actions, making robustness to errors in that information particularly relevant. We test RAP under two forms of uncertainty: changes to the robot's initial pose and changes to the visual environment.

Action-level uncertainty. Following A2A (Jia et al., 2026), we vary the initial robot pose to test robustness when the required actions and subsequent histories or previews differ from the demonstrations. In simulation, we perturb the initial joint angles by 0.01, 0.02, 0.04, or $0.08\,\mathrm{rad}$. Figure 6 compares success across these levels on three tasks, grouping methods by whether their source contains sampled noise. We also include N-A2A (Jia et al., 2026), which injects Gaussian noise with standard deviation 0.1 into A2A's historical-action source.

At the largest perturbation, three-NFE RAP exceeds A2A and VITA by 22–50 percentage points. SG-RST trains on the policy's own imperfect previews, whose errors vary across demonstration windows and policy updates. Exposure to these diverse source errors may help RAP learn robust preview refinement without additional Gaussian noise. Section 5.2 further examines whether this training construction improves source alignment between training and deployment.

Visual uncertainty. We next test whether improved action-level robustness is accompanied by a loss of visual robustness. Table 2 evaluates progressively stronger changes to backgrounds, lighting, and camera poses; Appendix A.6 gives the protocol.

RAP retains visual robustness comparable to A2A, with both reaching 25.3% average OOD success at six NFEs. N-A2A performs worse despite its improvement under initial pose uncertainty, suggesting a trade-off for the evaluated noise configuration. These results support learning from policy-generated source errors as a way to improve action-level robustness while retaining visual performance.

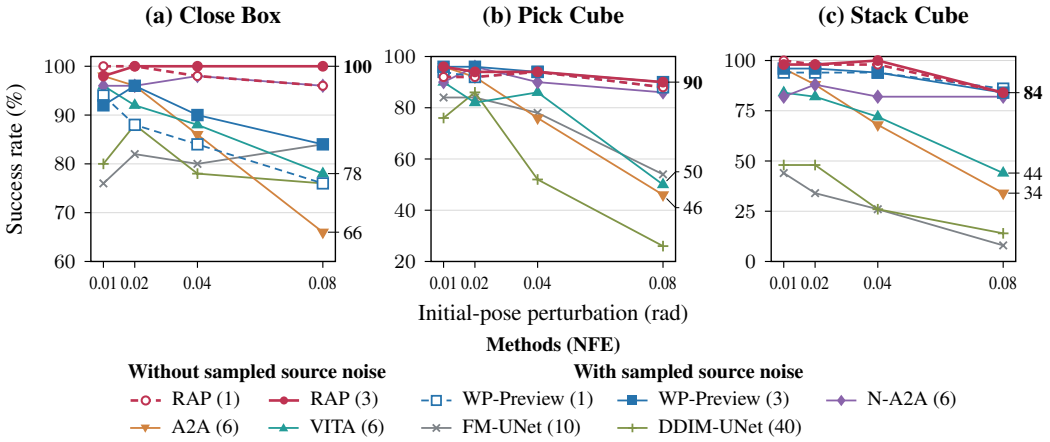


Figure 6: **Robustness to initial pose uncertainty.** Each point reports success over 50 rollouts after 50 training epochs. At 0.08 rad, the best RAP result exceeds the two baselines without sampled source noise, A2A and VITA, by 22–34, 40–44, and 40–50 percentage points on Close Box, Pick Cube, and Stack Cube, respectively.

Table 2: Close Box success rate (%) under visual uncertainty, after 200 training epochs and using 50 rollouts per cell. L0 is in-distribution; L1 changes backgrounds, L2 additionally changes lighting, and L3 additionally changes camera poses. OOD Avg. averages L1–L3. Bold marks the best result in each OOD row, including ties.

Condition	Method	RAP			WP-Preview			A2A		N-A2A		VITA	FM-UNet	DDIM-UNet
	NFE	1	3	6	1	3	6	1	6	1	6	6	10	40
L0 (ID)		100	100	100	100	100	100	100	100	100	100	100	94	94
L1		20	28	28	2	**32**	20	22	30	18	18	2	0	2
L2		18	20	**26**	0	24	16	18	**26**	12	24	2	4	4
L3		20	16	**22**	0	16	10	20	20	16	20	0	2	4
OOD Avg.		19.3	21.3	**25.3**	0.7	24.0	15.3	20.0	**25.3**	15.3	20.7	1.3	2.0	3.3

5.2 Source Consistency

We test whether SG-RST makes training sources closer to those used at deployment, using the same 98 recorded Close Box training trajectories for both policies. To obtain deployment sources, we run each policy from the start of a trajectory, passing its predicted preview to the next decision. At the same decision points, we separately construct the sources used during training: expert-action sources for WP-Preview, and previews generated by two preceding policy calls for RAP. We compare each training source with the corresponding deployment source.

For each policy, we compute the root-mean-square difference between the training and deployment sources for the same observation sequence in normalized action space, excluding padded action steps (Eq. 17, Appendix B). This is the *paired source gap*. Both sources use the same recorded observations; WP-Preview also uses the same Gaussian perturbation for both. This isolates the difference caused by source construction from differences in observations or sampled noise.

Averaging within each trajectory and then across the 98 trajectories gives mean paired gaps of 0.0501 for WP-Preview and 0.0181 for RAP, a relative reduction of 64%. RAP has a smaller trajectory-mean gap on 96 of 98 demonstrations (Figure 7(a)). The medians are similar, but RAP reduces the large gaps in the upper tail (Figure 7(b)). These results support the two-call training approximation: it produces previews closer to those obtained by running the policy from the trajectory start.

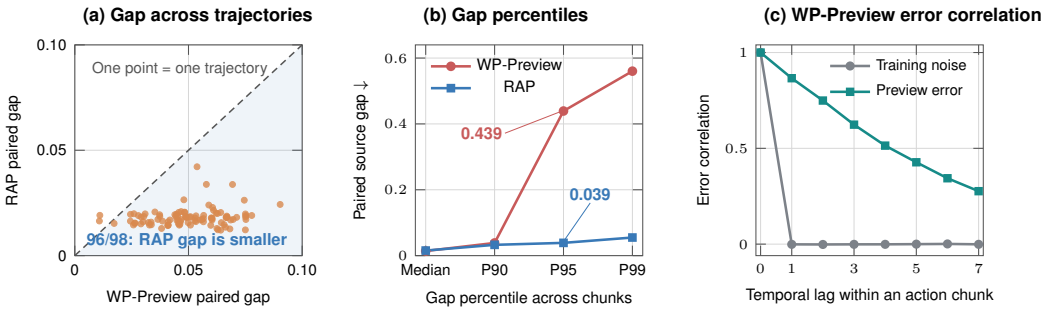


Figure 7: **Source consistency on recorded Close Box observations**, using three-NFE policies trained for 30 epochs. **(a)** Each point compares trajectory-mean paired gaps for one demonstration; points below the diagonal favor RAP. **(b)** Percentiles weight trajectories equally; P95 marks the threshold exceeded by the largest 5% of gaps. **(c)** WP-Preview's deployment preview errors are temporally correlated, whereas its Gaussian training noise is nearly uncorrelated.

WP-Preview's deployment preview errors are temporally correlated, unlike its nearly uncorrelated Gaussian training noise (Figure 7(c)). This supports SG-RST's use of policy-generated previews to learn their temporal error structure. Appendix B provides further distribution-level diagnostics.

5.3 Ablation Studies

Table 3 varies training depth J and, with $J = 2$ fixed, compares TVF with the current eight-frame observation window (Single8) and a continuous 16-frame window (Single16).

Table 3: Success rate (%) for component ablations after 30 training epochs, using 100 rollouts per cell. Bold marks the best result within each ablation block.

Variant	Task	Close Box		Pick Cube		Stack Cube	
	NFE	1	3	1	3	1	3
Source exposure depth							
$J = 1$		61	72	66	78	10	3
$J = 2$		**89**	**97**	**100**	97	**97**	94
$J = 3$		68	80	98	**100**	89	**100**
Visual condition with $J = 2$							
TVF		89	**97**	**100**	**97**	**97**	**94**
Single8		**95**	81	97	92	89	83
Single16		92	87	98	90	74	91

Recursive training depth. Increasing J from 1 to 2 improves all six settings. The extra call reuses a predicted preview before producing the training source, better matching deployment. We use $J = 2$, since a third call increases training cost with limited additional benefit.

Temporal visual fusion. TVF outperforms both alternatives in five of six settings. The Single16 comparison supports retaining the preview's original visual feature alongside current feedback.

6 Conclusion

We introduced RAP to address train–deployment source mismatch in temporal-source visuomotor policies. RAP refines unexecuted action previews through latent flow matching. SG-RST recursively generates these previews with the current policy and supervises their refinement using offline demonstrations, aligning source construction with deployment without additional Gaussian noise. TVF guides refinement with cached and current visual features, avoiding repeated encoding of past observations. Experiments validate RAP's fast action generation and robustness under action-level and visual uncertainty.

AI USE STATEMENT

Generative AI tools were used to assist with language polishing and refinement of wording in the manuscript. The authors reviewed all AI-assisted revisions and take full responsibility for the paper's content.

ETHICS STATEMENT

The study uses manipulation benchmarks and task-specific robot demonstrations. Physical experiments involve controlled tabletop manipulation on a Franka arm. Deployment beyond these laboratory tasks requires application-specific assessment of robot safety and failure recovery.

REPRODUCIBILITY STATEMENT

Sections 4 and 5 describe the source construction, update order, objectives, tasks, and main evaluations. Appendices A–D provide implementation settings, evaluation tables, and pseudocode. Appendix A reports the network and training configuration, and Appendix A.4 specifies the fixed-epoch checkpoint protocol.

REFERENCES

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*, 2025. doi: 10.15607/RSS.2025.XXI.010.

Kaiqi Chen, Eugene Lim, Kelvin Lin, Yiyang Chen, and Harold Soh. Don't start from scratch: Behavioral refinement via interpolant-based policy diffusion. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.122.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10–11):1684–1704, 2025. doi: 10.1177/02783649241273668.

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. *arXiv preprint arXiv:2606.17408*, 2026.

Dechen Gao, Boqi Zhao, Andrew Lee, Ian Chuang, Hanchu Zhou, Hang Wang, Zhe Zhao, Junshan Zhang, and Iman Soltani. VITA: Vision-to-action flow matching policy. In *International Conference on Learning Representations*, 2026.

Zifeng Gao, Guanxing Lu, Tianxing Chen, Wenxun Dai, Ziwei Wang, Chao Shang, Wenbo Ding, and Yansong Tang. ManiCM: Real-time 3D diffusion policy via consistency model for robotic manipulation. *arXiv preprint arXiv:2406.01586*, 2024.

Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, Yutong Liang, Dylan Goetting, Chaoyi Xu, Haozhe Chen, Yuxi Qian, Yiran Geng, Jiageng Mao, Weikang Wan, Mingtong Zhang, Jiangran Lyu, Siheng Zhao, Jiazhao Zhang, Jialiang Zhang, Chengyang Zhao, Haoran Lu, Yufei Ding, Ran Gong, Yuran Wang, Yuxuan Kuang, Ruihai Wu, Baoxiong Jia, Carlo Sferrazza, Hao Dong, Siyuan Huang, Yue Wang, Jitendra Malik, and Pieter Abbeel. RoboVerse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning. *arXiv preprint arXiv:2504.18904*, 2025.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark and learning environment. *IEEE Robotics and Automation Letters*, 5(2): 3019–3026, 2020. doi: 10.1109/LRA.2020.2974707.

Jindou Jia, Gen Li, Xiangyu Chen, Tuo An, Yuxuan Hu, Jingliang Li, Xinying Guo, and Jianfei Yang. Action-to-action flow matching. In *Proceedings of Robotics: Science and Systems*, 2026. doi: 10.15607/RSS.2026.XXII.209.

Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. In *Proceedings of the 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 238–257. PMLR, 2025.

Sinjae Kang, Chanyoung Kim, Kaixin Wang, Li Zhao, and Kimin Lee. WarmPrior: Straightening flow-matching policies with temporal priors. *arXiv preprint arXiv:2605.13959*, 2026.

Jinhao Li, Yuxuan Cong, Yingqiao Wang, Hao Xia, Shan Huang, Yijia Zhang, Ningyi Xu, and Guohao Dai. STEP: Warm-started visuomotor policies with spatiotemporal consistency prediction. *arXiv preprint arXiv:2602.08245*, 2026a.

Shaolong Li, Lichao Sun, and Yongchao Chen. One-step flow policy: Self-distillation for fast visuomotor policies. *arXiv preprint arXiv:2603.12480*, 2026b.

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

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1939.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2022.

Dylan Miller and Martin Jagersand. Temporal policy: History-initialized action generation for robotic learning from demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2026.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.071.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=St1giarCHLP`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. FlowPolicy: Enabling fast and robust 3D flow-based policy via consistency flow matching for robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 14754–14762, 2025. doi: 10.1609/aaai.v39i14.33617.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.016.

A Implementation Details and Hyperparameters

Table 4 summarizes the evaluated configuration. Table 5 gives the network modules, adapted from the action-latent architecture of A2A (Jia et al., 2026). The following subsections explain how the inputs are constructed, how the losses train the policy, and how training and evaluation are performed.

Table 4: RAP configuration. Source construction and loss weights are detailed in Appendices A.1 and A.2, respectively.

Item	Setting
Executed actions per decision	8
Predicted actions per call	16: 8 current actions and an 8-action preview
Frames per observation window	8
Source construction	Cached preview followed by 8 copies of its final action
Initialization without a preview	Repeat the action corresponding to the current robot state to fill the source
Sampled source noise	None
Visual condition	Fuse the cached previous-window feature with the current-window feature
Recursive training depth	At most 2 preceding predictions with gradients stopped; update the policy through the current prediction
Action latent dimension	512
Euler steps per prediction	1 or 3 in the main comparison; also 6 in the Close Box visual-uncertainty comparison
Simulation evaluation	100 rollouts per cell for nominal results and ablations; 50 for each uncertainty condition

Table 5: RAP network modules, based on A2A's architecture (Jia et al., 2026). Source and target actions use the same action encoder.

Component	Implementation
Visual encoder	ResNet-18 extracts a feature for each frame; the window projector maps the concatenated 8-frame features to a 512-dimensional vector
Temporal visual fusion	Concatenate the cached and current 512-dimensional window features, then apply a linear projection from 1024 to 512 dimensions
Action encoder E_a	Three one-dimensional convolutional layers with kernel size 5, followed by a projection to 512 dimensions; each input contains 16 actions
Flow network v_θ	SimpleFlowNet with four MLP blocks using adaptive layer normalization; the latent input, visual condition, and output are each 512-dimensional
Action decoder D_a	Input projection, three MLP blocks, and an output projection that produces all coordinates of the 16 predicted actions

The encoder, vector field, and decoder are trained jointly. The shared action encoder maps both the source sequence and the expert target into the same latent space. The decoder receives two kinds of input during training: an encoded expert target and a latent endpoint predicted by flow integration. Their respective losses are explained in Appendix A.2. The preceding recursive calls use these same modules, but their outputs are detached before the supervised call.

A.1 Source construction

What one prediction contains. A decision cycle is one policy query followed by the execution of an action chunk. We index these cycles by k. Each executed chunk contains $H = 8$ actions, so consecutive decisions occur H control steps apart: $t_{k+1} = t_k + H$. The policy predicts two chunks at once. Let A_k be the expert actions for the current execution interval and A_{k+1} those for the following interval. The training target and policy prediction are

$$Y_k = [A_k; A_{k+1}], \qquad \hat{Y}_k = [\hat{A}_k; \hat{P}_{k+1}] \in \mathbb{R}^{2H \times d}, \tag{7}$$

where d is the number of coordinates in one action and the semicolon joins sequences along time. Only the first half, $\hat{A}_k$, is executed. The second half, $\hat{P}_{k+1}$, is saved as the preview for the next decision. Throughout this appendix, action sequences entering or leaving the network are in normalized action coordinates.

How a cached preview becomes the source. At decision k, the cached preview $P_k = \hat{P}_k$ comes from the preceding policy prediction. It already covers the current execution interval, but contains no actions for the next interval. We extend it by repeating its final action H times:

$$S_k = \mathcal{R}(P_k) = [P_k; \mathrm{RepeatLast}(P_k, H)]. \tag{8}$$

Here $\mathcal{R}$ denotes source extension, and $\mathrm{RepeatLast}(P_k, H)$ is the sequence of H identical copies of the last action in P_k. The first half of S_k corresponds in time to A_k; its repeated tail corresponds to A_{k+1}. For example, an 8-action preview becomes a 16-action source whose last 8 entries all equal the preview's final action.

This extension provides a constant-action initialization for the newly introduced future interval. For absolute-position actions, it initially keeps the predicted final configuration unchanged over that interval. Flow matching can refine both halves, so the repeated tail does not force the final prediction to remain stationary. It also gives the shared encoder the same 16-action input length for the source and the expert target. The purpose is to define the missing future actions and their temporal correspondence; the common latent dimension alone would not determine how to fill that interval.

How a prediction starts without a cached preview. At the start of an episode, no previous policy prediction is available. We therefore construct an action from the current robot state q_0, normalize it in the same way as the training actions, and repeat it to initialize the preview:

$$b(q) = \mathcal{N}_A(\mathcal{T}_A(q)), \qquad P_0 = \mathrm{Repeat}(b(q_0), H). \tag{9}$$

The map $\mathcal{T}_A$ expresses the robot state in the task's action coordinates, and $\mathcal{N}_A$ is the action normalizer fitted to the training data. The notation $b(q)$ abbreviates these two operations. In joint-position control, the resulting command requests the current joint configuration. Repeat makes H copies of this command; applying $\mathcal{R}$ then gives a full $2H$-action source with the same command at every step.

Each short training recursion starts in the same way, using the robot state recorded at its first observation window. After a prediction, the preview is already normalized and is cached in that form. Only the actions sent for execution are unnormalized. No Gaussian noise is added to either initialization.

A.2 Learning objectives

RAP follows A2A's joint training of the action encoder, vector field, and decoder (Jia et al., 2026). The losses serve three purposes: learn the latent transport, preserve action information in the latent representation, and make the finite-step prediction agree with the expert target. We describe them for the current supervised call; preceding calls only construct its source.

Let $z_k^S = E_a(S_k)$ and $z_k^Y = E_a(Y_k)$ be the encoded source and expert target, respectively. These are the same two encodings as in Eq. 4. The visual condition c_k is obtained by TVF from the cached and current visual features. The following equations describe one supervised example, with losses aggregated over the training minibatch.

Learning the latent flow. Following conditional flow matching (Lipman et al., 2023), we sample one flow time t uniformly from $[0, 1]$ and interpolate between the two encodings: $z_{k,t} = (1-t)z_k^S + tz_k^Y$. The vector field is trained to predict the displacement from source to target:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1)}\left[\|v_\theta(z_{k,t}, t, c_k) - (z_k^Y - z_k^S)\|_2^2\right]. \tag{10}$$

This loss uses one vector-field evaluation at the sampled point.

Preserving action information. The autoencoder loss reconstructs the expert sequence directly from its encoding:

$$\mathcal{L}_{\mathrm{AE}} = \|D_a(z_k^Y) - Y_k\|_1. \tag{11}$$

It trains E_a and D_a to preserve the action sequence.

Supervising the prediction used at inference. The decoder must also work with the endpoint produced by the learned flow, which need not be identical to the encoded expert target. We therefore run the same K-step Euler solver used at deployment:

$$\hat{z}_k^{(K)} = \mathrm{Euler}_K(z_k^S; c_k). \tag{12}$$

For use in the pseudocode, a complete policy prediction is written as

$$\hat{Y}_k = \mathrm{RAP}_\theta^{(K)}(S_k, c_k) = D_a(\mathrm{Euler}_K(E_a(S_k); c_k)). \tag{13}$$

A2A's inference-consistency objective supervises this single predicted endpoint in two spaces:

$$\mathcal{L}_{\mathrm{lat}} = \|\hat{z}_k^{(K)} - z_k^Y\|_1, \tag{14}$$

$$\mathcal{L}_{\mathrm{rec}} = \|D_a(\hat{z}_k^{(K)}) - Y_k\|_1, \tag{15}$$

$$\mathcal{L}_{\mathrm{IC}} = \mathcal{L}_{\mathrm{lat}} + \lambda_{\mathrm{act}}\mathcal{L}_{\mathrm{rec}}. \tag{16}$$

The first term brings the predicted endpoint closer to the expert encoding; the second checks whether decoding that endpoint produces the expert actions. Both terms use the same K-step integration. Direct action supervision through the solver connects the learned flow to its decoded output, helping prevent latent action space collapse during joint training (Jia et al., 2026; Gao et al., 2026). Both action-space reconstruction losses cover all $2H$ actions, including the next preview.

Combining the losses and updating the policy. Equation 6 combines the flow-matching, autoencoder, and inference-consistency objectives. The last objective contains the latent and action terms above, so there are four constituent losses. We use $\lambda_{\mathrm{FM}} = 1$, $\lambda_{\mathrm{AE}} = 0.5$, $\lambda_{\mathrm{IC}} = 1$, and $\lambda_{\mathrm{act}} = 0.5$, following A2A (Jia et al., 2026).

Gradients from the current supervised call update the shared action encoder, visual encoder, TVF projection, vector field, and decoder. The visual encoder receives gradients through the current observation window, and the endpoint losses propagate through all K Euler steps. The preceding predictions and cached visual features are detached: they are inputs to the current update, but gradients do not continue into the calls that generated them. The next training examples construct fresh previews using the updated policy.

A.3 Training configuration

Recursive training cost. Two counts are needed to describe the training procedure. The recursion depth J counts preceding policy calls; the solver budget K counts vector-field evaluations within each call. One call with $K = 3$, for example, evaluates the vector field at three Euler steps to produce one action sequence. RAP uses a maximum depth of $J = 2$ and trains a separate policy for each deployment budget.

For a training example with two preceding cycles available, the vector field is evaluated in three parts:

1. **Construct the source: $2K$ evaluations.** Two preceding policy calls generate and reuse a preview. Each call uses K Euler steps, and gradients through both calls are stopped.
2. **Predict the supervised endpoint: K evaluations.** The current call integrates from its constructed source. The latent-consistency loss and decoded-action loss share this endpoint, so their combined cost is K evaluations, not $2K$.
3. **Evaluate the flow-matching loss: one evaluation.** A separate call to the vector field is made at a randomly sampled interpolation point and flow time. The autoencoder loss needs no vector-field call.

Thus, a full-depth example uses $2K + K + 1 = 3K + 1$ vector-field forward evaluations: 4 when $K = 1$, and 10 when $K = 3$. This counts calls to the vector field, not all operations or elapsed training time. Visual encoding, action encoding and decoding, and the backward pass are additional computations. In particular, a detached call and a differentiable call do not have the same backward-pass cost.

At the beginning of a demonstration, fewer preceding cycles are available. If k counts cycles from zero, the actual number of preceding calls is $j_k = \min(J, k)$. The same accounting gives $j_k K + K + 1$

evaluations: $K+1$ for the first cycle, $2K+1$ for the second, and $3K+1$ thereafter. Comparisons use the same number of training epochs, while this source-generation work is specific to RAP's training procedure. At deployment, the preview is already cached, so a decision uses only the current K-step prediction.

Visual windows. The evaluated configuration uses an observation window of $m = 8$ frames and an execution chunk of $H = 8$ actions. At decision k, the window $\mathcal{O}_k$ contains the m frames ending at control step t_k. The visual encoder processes this window once. TVF combines its feature with the feature saved at the preceding decision, avoiding another encoder pass over the earlier window. The current feature is then cached for the next decision.

A.4 Evaluation protocol

Tasks and action representation. Simulation tasks are run through the RoboVerse infrastructure (Geng et al., 2025). Close Box, Pick Cube, and Stack Cube use Isaac Sim; the remaining tasks use MuJoCo (Todorov et al., 2012). Simulation actions are represented as joint angles.

Baseline configurations. A2A (Jia et al., 2026) constructs its source from executed-action history. WP-Preview (Kang et al., 2026) uses an expert forecast proxy during training and a predicted preview at deployment, with Gaussian residual noise fixed at $\sigma = 1.0$. Our WP-Preview implementation uses the same TVF condition as RAP. VITA (Gao et al., 2026) starts from a visual latent, while FM-UNet (Lipman et al., 2023) and DDIM-UNet (Chi et al., 2025; Song et al., 2021) start from Gaussian noise. In the robustness experiments, N-A2A adds i.i.d. Gaussian noise with standard deviation 0.1 to A2A's normalized historical-action source, following its initial-state-uncertainty configuration (Jia et al., 2026).

For the nominal comparison, RAP and WP-Preview use one or three NFEs, A2A and VITA use six, FM-UNet uses ten, and DDIM-UNet uses forty. Each RAP solver budget is trained separately and used consistently during recursive prediction, supervised integration, and deployment. Appendix A.3 details RAP's training computation.

Table 6: Task-specific data and training budgets. A dash denotes an unevaluated setting.

Task	Benchmark	Backend	Demos	Table 1 epochs	Perturbation epochs	Table 2 epochs
Close Box	RLBench	Isaac Sim	100	30	50	200
Pick Cube	ManiSkill	Isaac Sim	100	30	50	–
Stack Cube	ManiSkill	Isaac Sim	100	30	50	–
Pull Cube	ManiSkill	MuJoCo	100	30	–	–
Open Drawer	LIBERO	MuJoCo	46	30	–	–
Pick-Place Bowl	LIBERO	MuJoCo	33	30	–	–
Square	robomimic	MuJoCo	200	30	–	–
Tool Hang	robomimic	MuJoCo	200	200	–	–

For each task, all methods use the same demonstration count and the training budget specified for that experiment in Table 6. We evaluate the final-epoch checkpoint for every method; intermediate checkpoints are not selected using evaluation success rates. Nominal and ablation results use 100 rollouts per method, task, and condition; initial-pose perturbation and visual-domain randomization results use 50.

Inference latency. Table 7 supplements the NFE comparison in Table 1 with measured policy inference time. We benchmark the same Close Box checkpoints without retraining, using one 256×256 RGB camera stream and robot-state inputs. A complete policy call includes visual encoding and action decoding and produces an $H = 8$ execution chunk; environment simulation is excluded.

Table 7: Close Box policy inference on an RTX 5090 at FP32 and batch size one. Latency includes visual encoding and action decoding, reported as mean ± SD of five run means (50 warm-up and 1,000 timed calls per run). Params counts inference modules; each call produces an $H = 8$ execution chunk.

Method	NFE	Params (M)	Latency (ms/chunk)
RAP	1	35.22	2.266 ± 0.012
RAP	3	35.22	2.820 ± 0.008
WP-Preview	1	35.22	2.276 ± 0.008
WP-Preview	3	35.22	2.824 ± 0.022
A2A	6	34.66	3.686 ± 0.033

RAP takes 2.266 ms per chunk at one NFE and 2.820 ms at three NFEs. The three-NFE model reduces latency by 23.5% relative to six-NFE A2A. At equal NFE, RAP and WP-Preview have nearly identical latency and the same inference parameter count, so RAP's higher success comes at a comparable online cost. Their decoders use three MLP blocks, whereas A2A's decoder uses four. These measurements quantify policy inference time; the wall-clock frequency of a deployed controller also depends on environment execution and other control-system costs.

A.5 Effect of the number of function evaluations

Table 1 compares RAP policies trained separately for one and three function evaluations. Three evaluations improve success on Close Box, Pull Cube, Pick-Place Bowl, and Square, whereas one evaluation performs better on the other four simulated tasks. Thus, neither budget is uniformly better across tasks. Under initial-pose perturbations, three evaluations improve the mean success rate by 1.0 percentage point for RAP and 2.5 points for WP-Preview, averaged over the three tasks and four perturbation levels in Figure 6.

Flow matching generates actions by numerically integrating a learned vector field (Lipman et al., 2023). Taking more integration steps is intended to follow that field more accurately, but task success also depends on what the field has learned. A more accurate approximation to its ODE solution does not by itself guarantee actions closer to the expert target or higher task success.

In RAP, the solver budget also enters training. A one-step model is supervised on the endpoint of a single Euler update, while a three-step model is supervised on the endpoint reached after three updates. The same budget is used in the preceding calls that generate training previews, so changing K also changes the sources presented during learning. The results therefore compare policies trained for different budgets, rather than isolate solver accuracy for a fixed policy. They show that RAP can perform well at either budget; the choice depends on measured task success together with the inference latency in Table 7.

A.6 Close Box visual uncertainty

All methods in Table 2 are trained for 200 epochs and evaluated using the hierarchical RoboVerse randomization protocol reported with A2A (Jia et al., 2026). Level 0 uses the in-distribution setting, and Level 1 randomizes environmental background textures. Level 2 adds lighting changes: primary DiskLight intensity ranges from 12,000 to 45,000, auxiliary SphereLight intensity from 5,000 to 20,000, and color temperature from 2,500 to 6,500 K; ambient presets also vary. Ceiling-light direction changes by up to $\pm 15^\circ$, and SphereLight position by $\pm 0.5\,\mathrm{m}$ laterally/longitudinally and $\pm 0.3\,\mathrm{m}$ vertically. Level 3 additionally perturbs camera extrinsics uniformly by $\pm 0.2\,\mathrm{m}$ laterally/longitudinally and 0–$0.1\,\mathrm{m}$ upward. The OOD average gives equal weight to Levels 1–3.

B Matched-Observation Source-Consistency Analysis

We measure how much a policy's training source differs from its deployment source when both are constructed using the same recorded observations. We also examine whether independent Gaussian perturbations resemble the errors in policy-generated previews. The following protocol expands the results in Section 5.2 and Figure 7.

B.1 Comparing Sources at the Same Decision Point

We use the existing three-NFE, epoch-30 Close Box checkpoints for WP-Preview and RAP, without updating their parameters. For each recorded demonstration, we feed the policy the observation windows in chronological order, making one prediction every $H = 8$ action steps. After each prediction, we keep its unexecuted preview and use it to construct the source for the next prediction. This sequential replay follows the deployment rule for preview reuse. We reset the preview cache only at the start of a demonstration.

At each decision point, we separately construct the source that the same policy would use during training. This produces a pair of sources for the same point in the same demonstration. Both constructions use the recorded observations; predicted actions update the preview cache but are not executed in an environment. Holding observations fixed lets us examine the effect of source construction without also changing the scenes seen by the policy. The two checkpoints use identical action normalization.

WP-Preview. The training source is the expert action chunk plus Gaussian noise. The deployment source is the preview from the preceding sequential prediction plus Gaussian noise. We use the *same current noise sample* for both members of each pair. Their difference therefore comes entirely from replacing the expert chunk with the predicted preview: the common noise cancels when the two sources are subtracted. Noise from earlier predictions can still affect the preview carried forward through the replay. We repeat the replay eight times per demonstration with different Gaussian samples to account for this variation.

RAP. The training source is regenerated using SG-RST with $J = 2$. Starting from the recorded robot state two decision cycles earlier, the policy makes a prediction, passes its preview to a second prediction, and uses the resulting preview as the current training source. The paired deployment source is the preview carried forward by the sequential replay from the beginning of the demonstration. Thus, we compare a preview generated from two preceding calls with one generated through continuous preview reuse. Neither source receives additional Gaussian noise, and the audit records one RAP replay per demonstration.

B.2 Measuring the Paired Source Gap

For each pair, we measure how much the source action values differ. At cycle k of trajectory e, let $P^{\text{train}}_{e,k}$ and $P^{\text{deploy}}_{e,k}$ denote the first H source actions from the training and deployment constructions, respectively. Each action has d normalized coordinates. Their root-mean-square difference is

$$g_{e,k} = \sqrt{\frac{1}{Hd}\sum_{h=1}^{H}\sum_{r=1}^{d}\left(P^{\text{train}}_{e,k,h,r} - P^{\text{deploy}}_{e,k,h,r}\right)^2}, \tag{17}$$

where h indexes the future action step and r indexes an action coordinate. Here, $H = 8$ and $d = 9$, so each gap summarizes differences across 72 normalized action values. For WP-Preview, this equation is evaluated separately for each stochastic replay realization.

We compare only the first H source steps. The remaining H steps are produced by repeating the last preview action; including them would count that endpoint discrepancy repeatedly. We also exclude the first two decision cycles, where the full two-call training history is unavailable, and the final cycles without a complete $2H$ expert target. The resulting comparison covers 2,023 decision points across 98 demonstrations used to train the policies.

A small paired gap means that the source supplied during training resembles the source supplied during sequential replay. It does not by itself measure action accuracy. For example, if RAP's short-history and sequential previews overestimate the same action coordinate by similar amounts, their difference is small even though both differ from the expert. WP-Preview uses the expert chunk as the center of its training source, so its paired gap instead measures the deployment preview's error relative to that expert chunk. We therefore use the paired gap to assess how closely training and deployment sources agree. Whether the resulting actions complete the task is measured by the separate policy-rollout experiments.

B.3 Summarizing Gaps Across Demonstrations

We summarize the paired gaps to determine how large the source mismatch is on average, whether large discrepancies occur at some decisions, and how sensitive the estimated average is to the demonstrations included in the analysis.

Average gap across demonstrations. Longer demonstrations contain more decision points. If every gap contributed equally to the overall average, a demonstration with twice as many decisions would have twice as much influence. We instead give each demonstration equal importance: first average its valid gaps, including all eight replays for WP-Preview, and then average the 98 demonstration means. Each demonstration therefore contributes $1/98$ of the final mean, regardless of its length. This weighting is used to summarize the measured gaps; it does not change policy training.

Figure 7(a) shows the two policies' mean gaps for each demonstration. RAP has the smaller mean on 96 of the 98 demonstrations, so its advantage occurs across most of the recorded trajectories. Averaging these demonstration means gives 0.0501 for WP-Preview and 0.0181 for RAP, a 64% relative reduction.

Typical gaps and occasional large gaps. The mean alone does not show whether a policy usually has small gaps but occasionally produces a much larger mismatch. Figure 7(b) therefore reports percentiles of the individual decision-point gaps. The median describes the middle of the distribution. The 90th and 95th percentiles (P90 and P95) describe its high-gap portion: approximately 90% and 95% of the weighted gaps fall below these thresholds, respectively.

We retain equal importance per demonstration when calculating these percentiles. Each demonstration receives the same total weight, divided equally among its valid gaps and, for WP-Preview, its eight replays. We then sort all gaps from smallest to largest and locate the thresholds using their accumulated weights. Thus, a long demonstration does not dominate the percentiles simply because it supplies more decisions.

The weighted medians are 0.0140 for WP-Preview and 0.0154 for RAP, indicating similar typical gaps. Their P90 values are 0.0385 and 0.0328, whereas their P95 values differ much more: 0.4392 and 0.0387. The median slightly favors WP-Preview, but RAP has much smaller gaps near the upper end of the distribution. These percentiles help explain why RAP has a lower mean despite the similar medians: its advantage is most apparent among decisions with large source mismatches.

Uncertainty in the estimated mean. The reported mean is estimated from a finite set of 98 demonstrations. A confidence interval indicates how sensitive this estimate is to which demonstrations contribute to it. We estimate this sensitivity by bootstrap resampling: draw 98 demonstrations with replacement, average their previously computed mean gaps, and repeat. The policies remain fixed during this calculation.

We resample whole demonstrations because nearby predictions reuse earlier previews, making their gaps dependent. Keeping a demonstration's gaps together preserves this dependence in the calculation. The middle 95% of the resampled means forms the reported 95% confidence interval, with endpoints at their 2.5th and 97.5th percentiles. The intervals are $[0.0469, 0.0532]$ for WP-Preview and $[0.0173, 0.0190]$ for RAP. Their narrow ranges relative to the difference between the two means support the stability of the average-gap comparison across these recorded demonstrations. These intervals describe uncertainty in the overall mean; the decision-point percentiles above describe how individual gaps vary.

One additional demonstration was held out from policy training. Its mean gaps are 0.0491 for WP-Preview and 0.0172 for RAP. We report this trajectory separately and exclude it from the 98-trajectory summaries.

B.4 Examining the Structure of Preview Errors

The paired gap measures the size of the source mismatch. To understand why Gaussian perturbations may not represent the errors encountered during preview reuse, we also compare two quantities from WP-Preview: the Gaussian noise added to the expert training chunk, and the deployment preview minus the corresponding expert chunk. The latter is measured before adding the current

Gaussian noise. Both quantities are 8×9 arrays, with one error value for each future step and action coordinate. The following statistics pool all valid chunks and stochastic realizations from the 98 training demonstrations.

Correlation across future steps. An error can persist over several actions in a predicted chunk. Figure 7(c) measures this by comparing errors at future steps separated by a given lag. For lag one, for example, we pair the error at step 1 with that at step 2, step 2 with step 3, and so on. We collect these pairs from all valid chunks, compute their Pearson correlation separately for each action coordinate, and average the correlations across coordinates. Larger lags use the same procedure with more widely separated steps. This calculation concerns the future steps *within* a chunk, rather than consecutive policy queries.

The adjacent-step correlation is -0.0012 for the Gaussian training noise and 0.8660 for the deployment preview errors. The former is close to zero, as expected for independently sampled perturbations. The latter indicates that preview errors often persist across neighboring future actions. Thus, matching the amount of noise alone would not reproduce this temporal structure.

Variation across the whole action chunk. We further examine whether the errors vary independently across the 72 action values or tend to follow a few shared patterns. We flatten each 8×9 error array into a 72-dimensional vector and compute its covariance matrix across samples. Its eigenvectors describe directions in which the errors vary together, and each eigenvalue measures the variance along one such direction. We divide the sum of the five largest eigenvalues by the sum of all eigenvalues to measure how much of the total variation is captured by five directions.

These five directions explain 7.8% of the Gaussian-noise variation and 92.0% of the deployment preview-error variation. Together with the temporal correlations, this shows that the observed preview errors have a coordinated structure that independent Gaussian perturbations do not reproduce. The percentages describe the *structure* of the variation, rather than the absolute error magnitude.

B.5 Comparing Unpaired Source Distributions

The paired gap preserves the correspondence between two sources at the same decision point. A complementary question is whether the collection of training sources can be distinguished from the collection of deployment sources once that correspondence is discarded. To compare these collections across different action chunks, we subtract the corresponding expert chunk from each source. We call the resulting difference a *source residual*. Here we use the realized source values, including WP-Preview's sampled noise; the preview-error analysis above excludes the current noise sample.

Classifier-based comparison. For each policy, we label its training-source residuals and deployment-source residuals as two classes and fit the same L2-regularized logistic regression classifier. Its inputs describe each residual's values, squared values, temporal averages of products between values one to three future steps apart, per-step root-mean-square magnitude, and the mean and standard deviation of each action coordinate over the chunk. These features allow the classifier to detect differences in both typical values and error structure. Feature scaling is fitted using only the classifier's training fold. All samples from a demonstration are assigned to the same fold, so neighboring chunks from a replay never appear on both sides of a classifier training–evaluation split.

We pool the predictions made on held-out folds and report the area under the receiver operating characteristic curve (AUC). An AUC near 0.5 means the classifier has little ability to tell the two collections apart; larger values indicate greater distinguishability. The AUC is 0.5349 for WP-Preview and 0.8597 for RAP. These values assess distinguishability with this classifier and feature set, rather than establishing equality or inequality under every possible distribution test.

Sliced-Wasserstein comparison. We also compare the residual collections without fitting a classifier. We project each 72-dimensional residual onto 256 shared, randomly sampled unit directions. Along each direction, we sort the projected training and deployment values and average the absolute differences between corresponding ranks. Averaging over directions gives the reported sliced-Wasserstein distance. When a collection exceeds 5,000 samples, we use a shared random subset of source pairs for this calculation. The distances are 0.0081 for WP-Preview and 0.0094 for RAP.

These distribution comparisons pool individual residual samples; the equal-demonstration weighting described above applies to the paired-gap means and percentiles.

Relation to the paired results. Both supplementary measurements give a different ordering from the paired gap. WP-Preview adds unit-standard-deviation Gaussian noise to both its expert-based training source and its preview-based deployment source. This noise is much larger than the typical preview discrepancy and can make the two residual collections difficult to distinguish. It remains present in the unpaired comparison, whereas using the same noise in a source pair removes it from the paired difference. RAP has no additional Gaussian noise, so a smaller but systematic difference between its short-history and sequential previews can remain distinguishable. A smaller paired gap therefore need not imply a smaller difference between the unpaired distributions.

The replay results support the specific conclusion that RAP reduces the mean discrepancy between corresponding training and deployment source inputs and suppresses large gaps under the same recorded observations. They also identify temporal structure in preview errors that is absent from independent Gaussian perturbations. The task-success experiments in Table 1 and Section 5.1 complement this controlled analysis by executing policy actions and allowing subsequent observations to change.

C Self-Generated Recursive Source Training

Algorithm 1 describes how to train on one decision point in an offline demonstration. The training example provides the current observation window, the expert target for the current and next action chunks, and up to two preceding observation windows with their recorded robot states. The preceding windows are used to generate a preview for the current call; the expert actions remain the targets for supervision.

Choosing where to start the recursion. We count decision cycles from $k = 0$ and use a maximum of $J = 2$ preceding calls. The available history determines which calls are needed:

- **First cycle ($k = 0$).** There is no preceding window. The current robot state initializes the source, and the current visual feature is used in both TVF inputs. The policy immediately makes the supervised prediction.
- **Second cycle ($k = 1$).** The policy first predicts from the episode's initial window using the state-based source. Its detached preview and visual feature are then used by the supervised call at the second window.
- **Later cycles ($k \geq 2$).** The policy starts two windows before the supervised decision, using the state recorded there. It predicts once, reuses that preview in a second prediction, and passes the second preview to the current supervised call. Figure 4 illustrates this case.

In the algorithm, $j_k = \min(J, k)$ is the number of preceding calls actually made, and $s = k - j_k$ is the cycle where this short recursion starts. The state q_i is recorded at decision i, sg stops gradients, and Last_H selects the final H actions of a prediction. At the first call of each short recursion, there is no cached visual feature from an earlier call, so TVF uses that call's visual feature twice. Each subsequent call combines the previous cached feature with its own current feature.

Generating the source and applying supervision. The preceding calls pass their previews and visual features forward with gradients stopped, and receive no loss of their own. At the current decision, the shared action encoder encodes the resulting source again; this encoding is part of the differentiable supervised call. The current visual feature and TVF condition are also computed for this call. The flow-matching, autoencoder, and inference-consistency losses then update the shared policy parameters, as detailed in Appendix A.2. Stopping gradients through the earlier predictions therefore does not freeze the action encoder used by the current prediction.

All calls use the same K-step Euler solver. After a parameter update, later training examples generate their preceding previews with the updated policy. Predicted actions are never executed during this procedure: every call uses the observation window recorded in the demonstration. At deployment, the cache is instead carried forward continuously as the robot acts and receives new observations.

SG-RST approximates this preview-reuse process with a short history that can be constructed from offline data. Appendix A.3 gives the vector-field evaluation count for each of the three episode positions.

Algorithm 1 Self-Generated Recursive Source Training with maximum depth $J = 2$

Require: Demonstration observations and robot states, supervised cycle $k \geq 0$, target Y_k, solver budget K, maximum depth $J = 2$

$j_k \leftarrow \min(J, k)$; $s \leftarrow k - j_k$
$P \leftarrow \mathrm{Repeat}(b(q_s), H)$
for $i = s, \ldots, k-1$ **do**
  $h_i \leftarrow \mathrm{sg}(\phi(\mathcal{O}_i))$
  if $i = s$ **then**
    $c_i \leftarrow \mathrm{TVF}(h_i, h_i)$
  else
    $c_i \leftarrow \mathrm{TVF}(h_{i-1}, h_i)$
  end if
  $\tilde{Y}_i \leftarrow \mathrm{sg}(\mathrm{RAP}^{(K)}(\mathcal{R}(P), c_i))$
  $P \leftarrow \mathrm{Last}_H(\tilde{Y}_i)$
end for
$h_k \leftarrow \phi(\mathcal{O}_k)$
if $j_k = 0$ **then**
  $c_k \leftarrow \mathrm{TVF}(h_k, h_k)$
else
  $c_k \leftarrow \mathrm{TVF}(h_{k-1}, h_k)$
end if
$S_k \leftarrow \mathcal{R}(P)$
$z_k^S \leftarrow E_a(S_k)$; $z_k^Y \leftarrow E_a(Y_k)$
Sample $t \sim \mathcal{U}(0,1)$; $z_t \leftarrow (1-t)z_k^S + tz_k^Y$
$\mathcal{L}_{\mathrm{FM}} \leftarrow \|v_\theta(z_t, t, c_k) - (z_k^Y - z_k^S)\|_2^2$
$\hat{z}_k^{(K)} \leftarrow \mathrm{Euler}_K(z_k^S; c_k)$
$\mathcal{L}_{\mathrm{lat}} \leftarrow \|\hat{z}_k^{(K)} - z_k^Y\|_1$
$\mathcal{L}_{\mathrm{rec}} \leftarrow \|D_a(\hat{z}_k^{(K)}) - Y_k\|_1$
$\mathcal{L}_{\mathrm{AE}} \leftarrow \|D_a(z_k^Y) - Y_k\|_1$
$\mathcal{L}_{\mathrm{IC}} \leftarrow \mathcal{L}_{\mathrm{lat}} + \lambda_{\mathrm{act}}\mathcal{L}_{\mathrm{rec}}$
Backpropagate Eq. 6 through the current supervised call only; update the policy parameters.

D RAP Inference at One NFE

Algorithm 2 gives the one-NFE decision cycle. The policy refines its cached preview using current and cached visual features, executes the first H actions, and retains the remaining H. The operators First_H and Last_H select these two halves, respectively. An episode start or invalid cache uses the state-based initializer. For $K > 1$, only the latent update changes to K Euler steps.

Algorithm 2 RAP inference (one-NFE configuration)

Require: Cache (P_k, h_{k-1}), current window $\mathcal{O}_k$, robot state q_k

$h_k \leftarrow \phi(\mathcal{O}_k)$
if episode start or invalid cache **then**
  $P_k \leftarrow \mathrm{Repeat}(b(q_k), H)$; $h_{k-1} \leftarrow h_k$
end if
$S_k \leftarrow \mathcal{R}(P_k)$
$c_k \leftarrow \mathrm{TVF}(h_{k-1}, h_k)$; $z_k^S \leftarrow E_a(S_k)$
$\hat{z}_k^{(1)} \leftarrow z_k^S + v_\theta(z_k^S, 0, c_k)$
$\hat{Y}_k \leftarrow D_a(\hat{z}_k^{(1)})$
$\hat{A}_k \leftarrow \mathrm{First}_H(\hat{Y}_k)$
$P_{k+1} \leftarrow \mathrm{Last}_H(\hat{Y}_k)$
Cache h_k; unnormalize $\hat{A}_k$ and execute the action chunk.