ONEROBO: TEMPORALLY STRUCTURED DISTRIBUTION MATCHING FOR ONE-STEP ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion and flow-matching based robot policies model continuous action distributions but incur inference latency from iterative sampling. Few step distillation reduce this overhead, yet existing methods overlook the within-chunk temporal structure and cross-chunk execution consistency required for robot control. We introduce OneRobo, a one-step distillation framework for robot policies that preserves temporal structure. To accommodate the distinct dynamics of arm motion and continuous gripper control, we adopt channel-specific structured generation: arm generation and its distillation directions are constrained to a low-frequency discrete cosine transform (DCT) subspace to suppress unnecessary temporal variations, while teacher supervision on gripper changes across timesteps promotes stability. To improve cross-chunk consistency, we incorporate aligned historical-action initialization and learnable adapters into training. These mechanisms form a differentiable action-generation pipeline, with distribution matching and auxiliary supervision jointly applied to the final structured outputs. Across 50 RoboTwin simulation tasks with FastWAM, OneRobo reduces action generation from ten steps to one, with only a 1.64% drop in average success rate relative to the multi-step teacher under the Easy setting and a $3.23\times$ inference speedup. Additional experiments on $\pi_{0.5}$ and multiple real-world tasks further demonstrate its broad effectiveness.

1 INTRODUCTION

Recent advances in Vision-Language-action Models (VLAs) and World Action Models (WAMs) have opened new avenues for robotic manipulation by integrating vision, language, and action Black et al. (2024); Ye et al. (2026b). Representative models, including $\pi_{0.5}$ Intelligence et al. (2025), SmolVLA Shukor et al. (2025), and FastWAM Yuan et al. (2026), use flow-matching action generators to map multimodal features to continuous action chunks. However, iterative sampling requires multiple network evaluations and remains a major source of inference latency Su et al. (2026); Luan et al. (2026). In real-time robotic systems, this latency affects not only action generation speed but also when new observations inform decisions and how successive action chunks are executed Black et al. (a). Reducing sampling iterations is therefore important for the efficient deployment of generative policies Prasad et al. (2024).

To reduce sampling costs, prior work has distilled generative robotic policies into few or one step generators through consistency distillation Prasad et al. (2024); Gao et al. (2024), distribution matching Wang et al. (2024); Jia et al. (2024), and progressive self-distillation Luan et al. (2026). FreqPolicy further incorporates frequency-domain consistency to account for the temporal structure of action trajectories Su et al. (2026). These advances establish a foundation for one-step policies, but evaluating whether a distilled policy preserves its teacher's behavior requires distinguishing **pointwise action accuracy** from **temporal fidelity**. Errors of comparable magnitude may appear as either smooth trajectory offsets or rapidly alternating perturbations, yielding similar mean squared error (MSE) but different patterns of direction reversal. Thus, **pointwise action error alone may obscure execution-relevant deviations in temporal dynamics.**

To examine this distinction, we evaluate both pointwise error and temporal behavior, as reported in Tab. 1. In some reduced-step configurations, action MSE remains low, whereas temporal-difference metrics and motion direction reversal rates deteriorate substantially. In one representative com-

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.

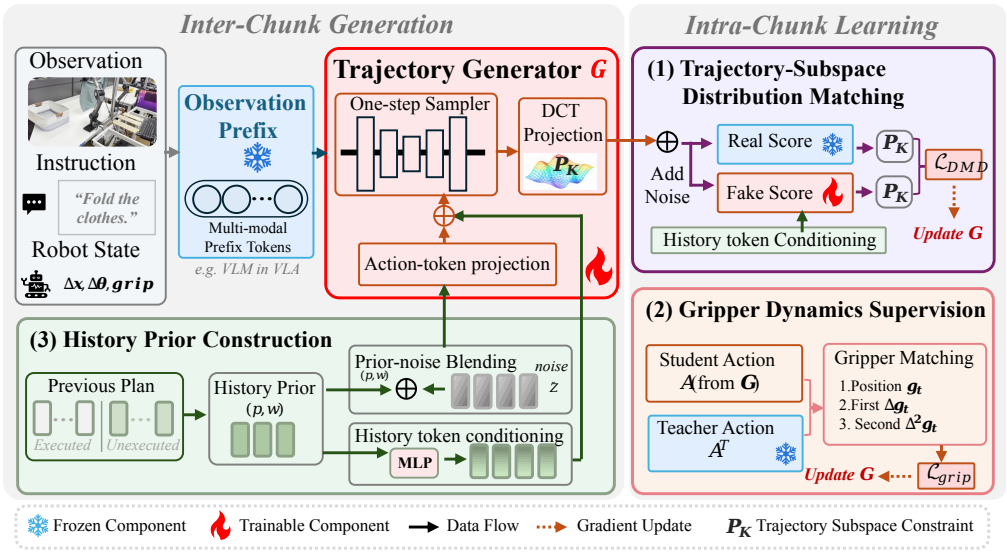


Figure 1: **Overview of OneRobo.** OneRobo integrates channel-aware temporal learning with history-conditioned generation. (1) combines DCT-constrained arm generation with teacher-guided distribution matching. (2) matches teacher values and temporal differences in control space. (3) incorporates the previous chunk's unexecuted actions through noise blending and token conditioning.

parison, the reversal rate increases from 5% to 26%. Fig. 2 illustrates this discrepancy: predicted trajectories remain close to the references but exhibit frequent local direction changes, which manifest as arm jitter in the execution tests. These observations show that low pointwise error does not necessarily imply preservation of the teacher's temporal dynamics.

This execution-relevant temporal structure has two interrelated aspects: intra-chunk dynamics and inter-chunk continuity. Within each chunk, arm and gripper actions have different requirements. Arm motion should preserve task-relevant trajectory changes while avoiding spurious high-frequency oscillations Su et al. (2026). Grippers must maintain stable states while allowing rapid opening and closing. Uniform smoothing may not satisfy both requirements. Across chunks, newly generated actions may disagree with the unexecuted portion of the previous plan. Such disagreements can occur in overlapping regions or at execution transitions, even when each chunk is internally well structured. RTC improves continuity during asynchronous execution by conditioning inference on overlapping actions from the previous chunk Black et al. (a). Subsequent work incorporates action-prefix conditioning into training to reduce inference-time correction overhead Black et al. (b); Liu et al. (2026). Building on these advances, we ask: **how can one-step generation and distillation jointly preserve channel-specific dynamics and inter-chunk continuity?** These aspects are coupled. Transforming a chunk can alter its overlap with the previous plan, while history conditioning shapes the trajectory within the new chunk.

To this end, we propose **OneRobo**, a **One**-step **Robo**tic Policy Distillation framework designed to preserve execution-relevant temporal structure. For arm actions, we construct a low-frequency trajectory subspace using a truncated discrete cosine transform (DCT) basis. We apply this constraint consistently to noise construction, action generation, and distribution-matching updates. For gripper actions, we retain the native representation and match the teacher's values and temporal differences. This supervision encourages stable holding and faithful opening and closing transitions without imposing the arm-specific low-frequency constraint. To improve inter-chunk continuity, we condition each new action chunk on the unexecuted portion of the previous plan and its validity. History conditioning is incorporated into both generation and distillation, allowing the student to use prior plans alongside current observations. Together, these mechanisms integrate channel-specific temporal modeling and cross-chunk continuation into a unified distribution-matching framework.

We evaluate OneRobo across multiple VLA and WAM models in simulation and on real robots. On all 50 RoboTwin tasks under the Easy setting, OneRobo distills Fast-WAM into a one-step policy with an average success rate only 1.64% below the ten-step teacher. It also achieves a 3.23× inference speedup. Our main contributions are as follows:

Table 1: Arm temporal metrics and action MSE.

Method	NFE	Arm D_1 ($\times 10^{-3}$)	Arm D_2 ($\times 10^{-3}$)	Direction reversal (%)	MSE ($\times 10^{-4}$)
Teacher	10	6.17	4.10	5.06	–
Teacher	5	6.78	5.32	8.21	0.31
Teacher	1	18.42	27.36	44.79	6.37
DMD	1	10.68	13.14	26.87	3.06
OneDP	1	8.80	9.19	14.64	**2.66**
FreqPolicy	1	10.02	11.58	19.21	3.68
OneRobo	1	**6.14**	**4.57**	**5.12**	2.71

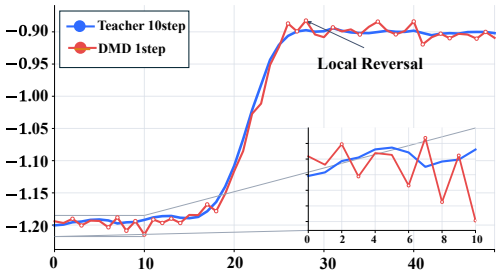


Figure 2: One-step DMD exhibits local oscillations and direction reversals.

1. We introduce **OneRobo**, a one-step distillation framework for VLA and WAM policies. We evaluate multiple models under various tasks to demonstrate the effectiveness.
2. We propose channel-aware intra-chunk modeling, combining trajectory-subspace distillation for arm actions with teacher-guided gripper supervision in control space.
3. We integrate history conditioning into one-step generation and distillation, jointly modeling intra-chunk dynamics and inter-chunk continuation without additional sampling steps.

2 Related Work

2.1 Generative Robotic Policies

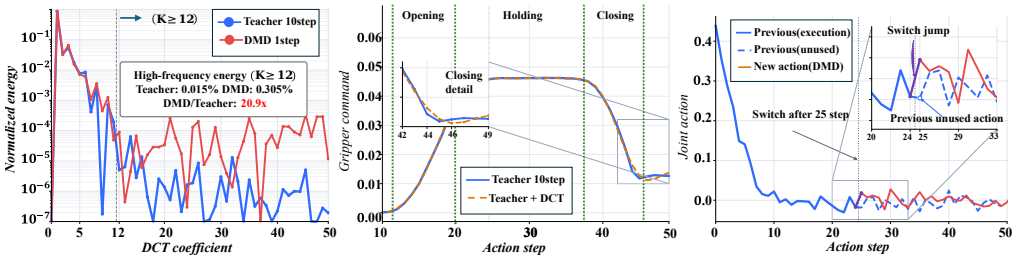
Diffusion and flow matching have become effective generative formulations for modeling multimodal continuous robot actions. Diffusion Policy introduced conditional denoising for action-sequence generation Chi et al. (2025), while RDT-1B scaled diffusion Transformers to large-scale bimanual manipulation Liu et al. (2025). More recently, VLAs such as π_0 Black et al. (2024), $\pi_{0.5}$ Intelligence et al. (2025), SmolVLA Shukor et al. (2025), and GR00T N1 Bjorck et al. (2025) combine pretrained vision-language representations with flow-based action generation to predict continuous action chunks. WAMs further incorporate visual dynamics into policy learning. DreamZero jointly models future video and robot actions using a pretrained video diffusion backbone Ye et al. (2026b). Fast-WAM Yuan et al. (2026) and GigaWorld-Policy Ye et al. (2026a) instead exploit future-video modeling as a training signal while avoiding explicit future-video generation during action inference, improving deployment efficiency. In contrast to redesigning the underlying policy architecture, we study how to distill the action generators of existing VLAs and WAMs into one-step policies.

2.2 Few-Step Action Generation

Reducing sampling steps is a standard approach to accelerating diffusion and flow models. Progressive Distillation recursively compresses multi-step samplers Salimans & Ho (2022), while Consistency Models enable one or few step generation by enforcing trajectory consistency Song et al. (2023). DMD instead trains a one-step generator by matching teacher and student distributions through score differences Yin et al. (2024b), with DMD2 further improving training stability and removing the need for pre-generated regression pairs Yin et al. (2024a). These techniques have increasingly been adapted to robotic policies. Consistency Policy distills Diffusion Policy through trajectory consistency Prasad et al. (2024), while OneDP Wang et al. (2024) and SDM Jia et al. (2024) Policy use distribution matching to obtain one-step action generators. For flow-based policies, OFP Li et al. (2026) learns one-step generation through self-distillation, whereas SnapFlow Luan et al. (2026) progressively distills VLAs. Flash-WAM Akbari et al. extends consistency distillation to WAMs. In contrast, we focus on preserving execution-relevant temporal structure during one-step distillation by jointly modeling channel-specific intra-chunk dynamics and inter-chunk history.

2.3 Temporal Action Structure and Inter-Chunk Continuity

Frequency-domain methods explicitly model temporal structure in action trajectories. Su et al. (2026) enforces frequency consistency across flow time, while FreqPolicy Zhong et al. (2026) progressively models hierarchical frequency components with continuous latent representations. In



(a) Excess High-Frequency Arm Energy (b) Gripper Dynamics under DCT Filtering (c) Inter-Chunk Replanning Inconsistency

Figure 3: Motivating observations. (a) Excess high-frequency arm energy in one-step DMD. (b) Gripper-transition deviations under DCT filtering. (c) Inter-chunk replanning inconsistency.

contrast, OneRobo constrains continuous arm trajectories to a shared low-frequency subspace, while retaining the gripper representation with teacher-guided supervision. Inter-chunk continuity has been addressed through previous-action conditioning. OFP Li et al. (2026) warm-starts one-step generation using the unexecuted previous action chunk. RTC Black et al. (a) maintains continuity through inference-time inpainting of overlapping actions, while training-time RTC Black et al. (b) directly learns action-prefix conditioning to remove this inference-time overhead. OneRobo instead incorporates history conditioning into both one-step generation and distribution-matching distillation, jointly modeling intra-chunk dynamics and inter-chunk continuation.

3 METHOD

OneRobo preserves the temporal structure of one-step robotic policies at both intra-chunk and inter-chunk levels. As shown in Fig. 1, trajectory subspace constraints and history conditioning regulate the generation process, while gripper supervision provides channel-specific behavioral fidelity.

3.1 PRELIMINARIES

Conditional Action Generation. Given an observation o, task instruction ℓ, and robot state s, the conditional policy generates an action chunk $A \in \mathbb{R}^{H\times D}$, where H and D denotes the prediction horizon and action dimension. We denote the conditioning information provided to the action expert as $c = \mathcal{E}(o, \ell, s)$. The policy outputs actions in a normalized action space. Following flow-matching policies, the action generator learns a velocity field $v_\eta(x_\tau, \tau; c)$ that transports samples from a noise distribution to the action distribution. During inference, generation starts from Gaussian noise $x_1 = z \sim \mathcal{N}(0, I)$. A conventional multi-step policy numerically integrates this ODE with multiple velocity evaluations to obtain the final action chunk, where τ denotes the flow time:

$$A = x_0 \approx x_1 - \int_0^1 v_\eta(x_\tau, \tau; c)d\tau. \tag{1}$$

OneRobo distills this iterative generation process into a single-step prediction:

$$A = z - v_\theta(z, 1; c). \tag{2}$$

Distribution Matching Distillation. Distribution Matching Distillation (DMD) introduces a distribution estimator (Fake Score) F_ϕ to approximate the student distribution, and uses the discrepancy between the Fake Score and Real Score distributions to provide the optimization direction for the generator G_θ. In OneRobo, we adopt DMD as the basic distillation framework. The teacher remains frozen, while G_θ and F_ϕ are optimized independently from the teacher initialization. The detailed distribution matching objective and optimization procedure are described in Sec. 3.4.

3.2 CHANNEL-AWARE INTRA-CHUNK MODELING

Different action channels exhibit distinct temporal dynamics within an action chunk. Therefore, instead of applying a unified temporal constraint to all channels, we introduce a truncated temporal trajectory subspace for arm motions and teacher-guided temporal supervision for gripper actions.

3.2.1 TRAJECTORY-SUBSPACE CONSTRAINT FOR ARM ACTIONS

Challenge 1a: Excess High-Frequency Variation in Arm Trajectories. When a policy directly predicts an entire action chunk in a single generation step, the absence of iterative refinement may introduce rapid changes between neighboring action positions. As shown in Fig. 2, these temporal oscillations appear as frequent direction reversals in joint trajectories. To further analyze this phenomenon, we transform the generated trajectories into the frequency domain. As shown in Fig. 3(a), the one-step DMD exhibits substantially increased energy in the high-frequency components. These excessive high-frequency variations are strongly associated with temporal oscillations in generated trajectories. However, directly smoothing the generated trajectories may suppress task-relevant motion patterns and cannot ensure that the generated actions follow the desired temporal structure.

To suppress unnecessary high-frequency variations while preserving the original action space, we constrain arm trajectory generation within a truncated temporal subspace. Specifically, we construct the subspace using the first K orthonormal DCT-II bases.

Let $B_K \in \mathbb{R}^{H\times K}$ contain the first K orthonormal DCT-II bases, including the DC component. The corresponding projection matrix is $P_K = B_K B_K^\top$. Given the arm action channel set $\mathcal{J}_{\text{arm}}$, we define a channel-wise projection operator as

$$[\mathcal{P}_K(X)]_{:,j} = \begin{cases} P_K X_{:,j}, & j \in \mathcal{J}_{\text{arm}}, \\ X_{:,j}, & j \notin \mathcal{J}_{\text{arm}}. \end{cases} \tag{3}$$

The student still predicts the original $H \times D$ action chunk, while arm trajectories are restricted to the selected temporal basis. This reduces the temporal degrees of freedom from Hd_{arm} to Kd_{arm} without changing the output dimension or introducing an additional DCT loss. Gripper and other channels remain unchanged.

Let $z \sim \mathcal{N}(0, I)$ denote the initial noise and (p, w) denote the history condition introduced in Sec. 3.3. Without history conditioning, $\tilde{z} = \mathcal{P}_K(z)$. The one-step generation is formulated as

$$A_\theta = \mathcal{P}_K\left[\tilde{z} - \mathcal{P}_K\left(v_\theta^G(\tilde{z}, 1; c, p, w)\right)\right]. \tag{4}$$

Eq. 4 requires only one velocity evaluation at $\tau = 1$. The projection $\mathcal{P}_K()$ is applied to the initialization, velocity, and final action, preventing high-frequency components from being introduced during one-step transport rather than performing smoothing only after generation. To incorporate this structural constraint into distillation, the projected arm noise and velocity are also used in F_ϕ training and DMD updates, as described in Sec. 3.4. The detailed subspace properties are provided in Appendix A.1.

3.2.2 CONTROL-SPACE TEMPORAL SUPERVISION FOR GRIPPER ACTIONS

Challenge 1b: Distorted Gripper Dynamics. Unlike arm trajectories, gripper actions contain discrete transition patterns, where high-frequency variations may correspond to meaningful opening and closing operations. Therefore, directly suppressing high-frequency components may remove task-relevant gripper behaviors.

To preserve gripper execution dynamics, we retain the original gripper representation and introduce teacher-guided temporal supervision . Specifically, during student optimization, we obtain an auxiliary action target A_T^{aux} from the frozen teacher and extract the gripper channels from both teacher and student actions, where $\mathcal{C}_{\text{dec}}$ denote the task-specific action decoder:

$$g_G = \text{Grip}\left(\mathcal{C}_{\text{dec}}(A_\theta; s)\right), \quad g_T = \text{Grip}\left(\mathcal{C}_{\text{dec}}(A_T^{\text{aux}}; s)\right). \tag{5}$$

A_T^{aux} is obtained from the frozen 10-step teacher under the same conditioning. We then supervise both gripper states and temporal variations using a Huber loss:

$$\mathcal{L}_{\text{grip}} = \sum_{r=0}^{2} \lambda_r \left\| D_r g_G - D_r g_T \right\|_{\text{Huber}}, \tag{6}$$

where D_r denotes the r-th order temporal difference operator and $\lambda_r \geq 0$ weights the corresponding loss term. The zeroth-order term matches gripper states, while the first- and second-order terms match the teacher's temporal transition dynamics. Detailed definitions of the difference operators, loss weights, and temporal weighting are provided in Appendix A.2.

Table 2: Simulation results on RoboTwin.

Method	Steps	Handover Mic	Open Microwave	Place_Can Basket	Place Fan	Put_Bottles Dustbin	Scan Object	Stack_Bowls Three	...	Avg.	Speedup
Easy (demo_clean)											
FastWAM	10	98	74	74	98	92	94	88		92.80	1.00×
FastWAM	1	48	8	56	58	16	48	60		72.08	3.59×
DMD2	1	72	20	64	82	24	80	78		85.08	3.43×
OneDP	1	74	**70**	74	**96**	**72**	**88**	78		89.16	3.21×
FreqPolicy	1	**98**	2	**78**	66	66	62	64		78.12	3.20×
OneRobo	1	96	64	76	92	64	**88**	**90**		**91.16**	3.23×
Hard (demo_randomized)											
FastWAM	10	98	70	66	94	86	88	94		91.96	1.00×
FastWAM	1	68	8	52	50	14	34	56		71.88	3.59×
DMD2	1	88	2	60	92	32	74	66		85.36	3.43×
OneDP	1	88	36	68	88	**62**	**82**	78		87.32	3.21×
FreqPolicy	1	92	32	**74**	66	40	58	68		79.24	3.20×
OneRobo	1	**94**	**56**	62	**96**	60	80	**82**		**89.72**	3.23×

3.3 History-Conditioned Inter-Chunk Generation

Challenge 2: Inconsistent Replanning Across Action Chunks. One-step policies generate complete action chunks independently based on current observations. As shown in Fig. 3(c), during receding-horizon execution, however, the unexecuted actions from the previous chunk may become inconsistent with newly generated actions, leading to temporal discontinuities during replanning.

Therefore, we incorporate the previous action plan as an explicit history condition during one-step generation. Let A^- denote the previous action chunk and s^- its corresponding robot state. Assume that the first E actions of A^- have already been executed. We first decode the previous action chunk, remove the executed prefix, and re-encode the remaining plan under the current state s:

$$p = \mathcal{C}_{\text{enc}}\left(\text{Shift}_E\left[\mathcal{C}_{\text{dec}}(A^-; s^-)\right]; s\right), \quad w_{j,:} = m_j \max\left(1 - \frac{j}{O}, 0\right), \tag{7}$$

where $\mathcal{C}_{\text{dec}}$ and $\mathcal{C}_{\text{enc}}$ denote the task-specific action decoder and encoder, respectively, and $\text{Shift}_E(\cdot)$ removes the executed prefix and aligns the remaining actions with the current action chunk. The aligned historical plan p is associated with a confidence weight tensor w, which controls the contribution of historical actions during generation. As shown in Eq. 7, where j denotes the temporal index within the action chunk, m_j indicates the validity of the historical action, and O controls the temporal decay length. Therefore, p provides the historical action prior, while w controls its contribution during generation.

The history condition is incorporated into the one-step generation through two complementary paths. First, it modifies the initial noise by blending the projected noise with the historical plan:

$$\tilde{z} = \mathcal{P}_K\left[(1 - \lambda_{\text{hist}} w) \odot \mathcal{P}_K(z) + \lambda_{\text{hist}} w \odot p\right], \tag{8}$$

where $z \sim \mathcal{N}(0, I)$ denotes the initial Gaussian noise, λ_{hist} controls the history strength, and $\mathcal{P}_K(\cdot)$ is the trajectory-subspace projection introduced in Sec. 3.2.1. Second, the history condition is encoded as learnable embeddings and injected into the generator and distribution estimator:

$$e^i_{\text{prior}} = pW^i_p + wW^i_w, \qquad i \in \{G, F\}, \tag{9}$$

where W^i_p and W^i_w are learnable projection matrices for the generator ($i = G$) and distribution estimator ($i = F$). The two networks maintain independent history embeddings, while the frozen teacher does not receive history conditions. The projected history embedding is added to each noisy-action token before the first Action DiT block.

During training, we bootstrap the historical plan using the student policy under the previous observation without history input. The generated plan is detached and only used to construct p and w. During deployment, the history condition is obtained from the previously generated and partially executed action chunk.

3.4 Joint Distillation and Optimization

Given the one-step student output A_θ, we follow DMD2 to train a distribution estimator F_ϕ (Fake Score) that models the current student distribution. Specifically, we construct the flow state using

Table 3: Results on LIBERO benchmark.

Method	Steps	Spatial	Object	Goal	Long	Avg.	Speedup
$\pi_{0.5}$							
$\pi_{0.5}$	10	98.60 ± 0.40	98.53 ± 0.23	97.40 ± 0.20	92.27 ± 1.50	96.70 ± 0.46	1.00×
$\pi_{0.5}$	1	98.73 ± 0.64	98.07 ± 0.61	**97.73** ± 0.42	91.93 ± 0.70	96.62 ± 0.49	1.41×
OneDP	1	**98.93** ± 0.23	97.67 ± 0.70	97.60 ± 0.69	**92.27** ± 0.31	96.62 ± 0.20	1.34×
FreqPolicy	1	98.27 ± 0.70	**99.20** ± 0.40	97.13 ± 0.42	92.00 ± 0.53	96.65 ± 0.10	1.35×
OFP	1	83.73 ± 1.50	64.73 ± 1.01	66.87 ± 0.23	47.13 ± 0.81	65.62 ± 0.36	1.35×
OneRobo	1	98.67 ± 0.81	98.47 ± 0.23	97.47 ± 0.42	92.07 ± 0.64	**96.67** ± 0.29	1.34×
SmolVLA							
SmolVLA	10	70.40 ± 0.57	85.20 ± 0.28	74.90 ± 1.56	35.00 ± 0.57	66.38 ± 0.32	1.00×
SmolVLA	1	**80.00** ± 0.85	82.70 ± 0.99	66.60 ± 0.28	19.00 ± 0.85	62.08 ± 0.18	4.96×
OneDP	1	79.10 ± 0.42	78.90 ± 0.14	68.90 ± 0.14	25.90 ± 0.99	63.20 ± 0.35	4.72×
FreqPolicy	1	79.00 ± 0.28	85.10 ± 0.42	66.30 ± 0.42	24.30 ± 1.27	63.67 ± 0.18	4.63×
OneRobo	1	75.10 ± 1.84	**90.70** ± 0.42	**72.20** ± 0.85	**31.50** ± 0.14	**67.38** ± 0.11	4.82×

stop-gradient student samples, where $sg(\cdot)$ means stopping gradient calculation:

$$x_\tau = (1-\tau)sg(A_\theta) + \tau\mathcal{P}_K(\epsilon), \quad \epsilon \sim \mathcal{N}(0, I). \tag{10}$$

$\mathcal{P}_K$ denotes the arm trajectory projection defined in Sec. 3.2.1. The distribution estimator is optimized by

$$\mathcal{L}_F = \mathbb{E}\left[\left\langle \left(\mathcal{P}_K[v_\phi^F(x_\tau, \tau; c, p, w)] - (\mathcal{P}_K(\epsilon) - \mathrm{sg}(A_\theta))\right)^2 \right\rangle_m\right]. \tag{11}$$

For updating the student, we estimate the clean actions from the frozen teacher and the distribution estimator. Both estimates are treated as stop-gradient targets. The DMD update direction is computed as below, where d is a per-sample normalization factor that stabilizes the update magnitude:

$$\begin{aligned} \hat{A}_T &= x_\tau - \tau\mathcal{P}_K[v_T(x_\tau, \tau; c)], \\ \hat{A}_F &= x_\tau - \tau\mathcal{P}_K[v_\phi^F(x_\tau, \tau; c, p, w)], \\ \hat{u}_{\mathrm{DMD}} &= \mathrm{sg}\left[\frac{\hat{A}_F - \hat{A}_T}{d}\right], \end{aligned} \tag{12}$$

Then we follow DMD2 to compute the DMD objective $\mathcal{L}_{\mathrm{DMD}}$. Finally, the student objective is shown below, where α balances the auxiliary gripper supervision $\mathcal{L}_G = \mathcal{L}_{\mathrm{DMD}} + \alpha\mathcal{L}_{\mathrm{grip}}$.

4 Experiments

4.1 Experimental Settings

We evaluate OneRobo on RoboTwin Chen et al. (2025) and LIBERO Liu et al. (2023) simulation benchmarks, and two real-world manipulation tasks. RoboTwin evaluation covers all 50 tasks under both Easy and Hard settings, with 50 episodes per task per setting. LIBERO evaluation covers 40 tasks across LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long, with 50 episodes per task per evaluation run. The real-world tasks are *Fork Pick-and-Place* and *Folding the clothes*, illustrated in Fig. 4. The former requires selecting a fork by color, grasping it, and placing it at the target location. The latter requires picking up, folding, and storing a garment. Each method is evaluated in 30 fork-task trials with varied positions of differently colored forks, and 10 folding trials using garments of two colors.

Models and benchmarks. We evaluate Fast-WAM on RoboTwin, $\pi_{0.5}$ and SmolVLA on LIBERO, and $\pi_{0.5}$ on the real robot. Baselines include the ten-step teacher (Teacher-10), the same teacher evaluated with one step without distillation (Teacher-1), a DMD2-based Yin et al. (2024a), OneDP Wang et al. (2024), FreqPolicy Su et al. (2026), and OFP Li et al. (2026), with model-specific coverage shown in the result tables. All one-step variants use one velocity-network evaluation per action chunk. Within each backbone, methods are evaluated with the same action representation, prediction horizon, executed prefix length, and control interface.

Implementation details. All simulation teachers use publicly released checkpoints. For Fast-WAM on RoboTwin, distillation uses 2,500 clean demonstrations (50 per task), a batch size of 128, and

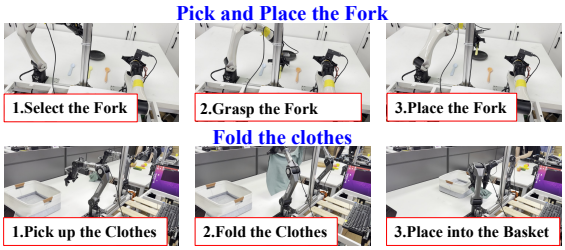


Figure 4: Real-world manipulation tasks.

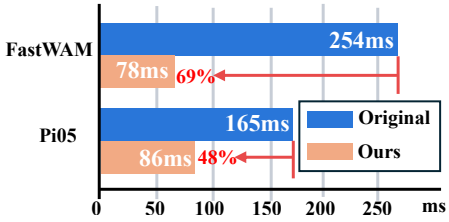


Figure 5: Inference efficiency comparison.

Table 4: Real-world task success on $\pi_{0.5}$.

Method	Steps	Fork Pick-and-Place	Garment Folding
Teacher	10	30/30	10/10
Teacher	1	21/30	0/10
DMD	1	24/30	2/10
OneRobo	1	**30/30**	**9/10**

Table 5: Component ablation of OneRobo.

Config.	Arm subspace	Gripper supervision	History conditioning	Success rate (%)
Exp0	–	–	–	79.2 (−2.4)
Exp1	✓	–	–	79.4 (−2.2)
Exp2	–	✓	–	78.6 (−3.0)
Exp3	–	–	✓	80.4 (−1.2)
Exp4	✓	✓	✓	**81.6**

5,000 training iterations on 4 A100GPUs. For $\pi_{0.5}$ and SmolVLA on LIBERO, we use 1,693 demonstrations, a batch size of 64, and 2,000 training iterations on 2 A100GPUs. All models are trained only on the action generator part. For example, $\pi_{0.5}$ trains only the action expert, and fastwam trains only ActionDiT, while the remaining parts are frozen, such as the VLM in pi05 and the Vadio DiT branch in fastwam. OneRobo retains $K = 16$ DCT basis vectors for Fast-WAM and SmolVLA, and $K = 3$ for the π_{05} LIBERO models. The default history blending strength is $\lambda_{\text{hist}} = 0.5$, and valid history conditions are retained with probability 0.9. The gripper supervision weights are $(\lambda_0, \lambda_1, \lambda_2) = (1, 0.5, 0.25)$. The two projections for p and w in Eq. equation 9 are each implemented as a single linear layer with the size `action_dim` $\times$ `hidden_dim`. For FastWAM, these two layers introduce only 0.028M additional parameters.

4.2 Simulation Results

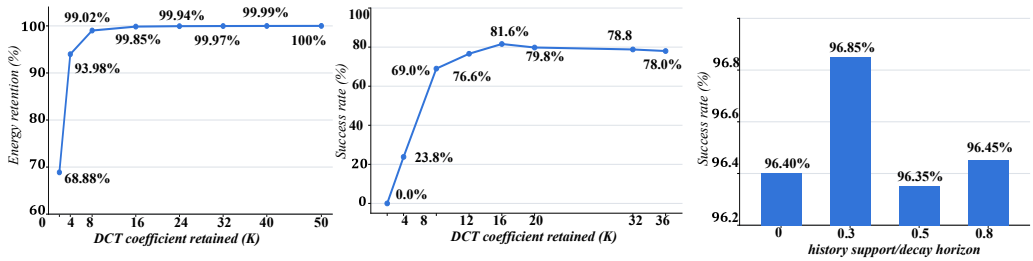
Fast-WAM on RoboTwin. Tab. 2 reports success rates averaged over all 50 tasks under both evaluation settings. Tab. 2 reports results on seven representative tasks, together with the average over all 50 tasks. Complete per-task results are provided in Appendix A.3. OneRobo achieves 91.16% success in the Clean setting and 89.72% in the Hard setting, compared with 92.80% and 91.96% for Teacher-10. Among the other trained one-step policies, OneDP obtains the highest overall success rate of 89.16% under Easy setting, which is still 2.00% lower than OneRobo. OneRobo performs better on long-horizon tasks that require relatively fine-grained manipulation, such as stack_bowls_three, nearly matching the teacher model's 10-step inference performance under Easy settings.

$\pi_{0.5}$ and SmolVLA on LIBERO. As shown in Table 3, OneRobo achieves an average success rate of 96.67% on $\pi_{0.5}$, closely matching the ten-step teacher at 96.70%. Teacher-1, OneDP, and FreqPolicy obtain 96.62%, 96.62%, and 96.65%, respectively. For SmolVLA, OneRobo achieves 67.38% average success, exceeding Teacher-10 by 1.00% and Teacher-1 by 5.30%. It also outperforms OneDP and FreqPolicy by 4.18% and 3.71%, respectively, and achieves the highest mean success among one-step variants on Object, Goal, and Long. Together, these results support OneRobo's cross-backbone applicability: it preserves near-teacher performance on $\pi_{0.5}$ and provides clearer gains on SmolVLA, where direct step reduction causes greater performance degradation.

Inference Speedup. As illustrated in Fig. 5, we show that on a single A100 GPU, FastWAM leveraging OneRobo reduces inference latency by 69%. On a single 4090 GPU, the latency of pi05 under PyTorch deployment can also be reduced from 165 ms to 86 ms, corresponding to a 48% reduction.

4.3 Real-World Evaluation

Table 4 reports real-world task success with $\pi_{0.5}$. On Pick-and-Place fork, OneRobo succeeds in all 30 trials, matching Teacher-10, compared with 21 successes for Teacher-1 and 24 for DMD. Teacher-1 exhibits noticeable arm jitter and mistimed gripper release. Observed failures include



(a) DCT Energy Retention (b) Success Rate of DCT Truncation (c) Success Rate of (H-E)/H

Figure 6: **Effect of DCT truncation and History-conditioning configuration.** (a) Trajectory energy retained with different numbers of DCT coefficients K. (b) Task success rate as K varies. (c) Task success rates under different History decay horizon.

unstable fork grasps and unsuccessful placements following premature or delayed gripper opening. DMD partially alleviates these issues but still exhibits unstable arm motion and gripper timing errors. OneRobo produces more stable execution and better-timed releases in the evaluated trials.

Folding the Clothes places greater demands on precise grasping and manipulation. For Teacher-1 and DMD, arm oscillations and gripper timing errors accompany incomplete movements and unstable garment grasps, preventing completion of the folding sequence in most trials. Teacher-1 succeeds in none of the 10 trials, while DMD succeeds in only two. In contrast, OneRobo mitigates these execution issues and completes 9 of 10 trials. These results demonstrate that OneRobo retains most of the teacher's observed task performance with one-step generation.

4.4 Ablation Studies

Component ablation. Table 5 reports a component ablation of OneRobo using SmolVLA on LIBERO-spatial. Exp0 uses the DMD baseline without the proposed components, Exp1–Exp3 each add one component from OneRobo, and Exp4 uses the full configuration. Starting from the DMD baseline at 79.2%, adding the arm-subspace constraint or history conditioning alone increases success to 79.4% and 80.4%, respectively. Gripper supervision alone yields 78.6%, which is 0.6 percentage points below the baseline. The full configuration achieves the highest observed success rate of 81.6%, improving over DMD by 2.4 percentage points.

DCT truncation. We train SmolVLA students with different numbers of retained DCT coefficients K and evaluate them on LIBERO-Spatial. Figure 6(a–b) examines energy retention and task success as the number of retained DCT coefficients K varies. Retaining only $K = 8$ coefficients captures 99.02% of the trajectory energy, increasing to 99.85% at $K = 16$. However, energy retention alone does not determine task performance. In the success-rate sweep, performance initially improves with K, peaking at 81.6% for $K = 16$, before decreasing to 78.0% at $K = 36$. Thus, retaining more frequency components does not necessarily improve execution, and an intermediate truncation level performs best among the tested settings.

History-conditioning configuration. We train $\pi_{0.5}$ students with different History-conditioning configurations and evaluate them on LIBERO-Spatial. Fig. 6(c) shows the accuracy when different proportions of the previous unexecuted actions are used as references for the current action. The highest success rate is achieved with the 0.3 configuration; that is, for π_{05} with a prediction length of 10, using the last 3 unexecuted actions from the previous step as the current reference yields the best performance.

5 Conclusion

We presented **OneRobo**, a one-step distillation framework for robotic policies. OneRobo combines arm trajectory-subspace constraints, teacher-guided gripper supervision, and history conditioning within distribution-matching distillation. Together, these components address intra-chunk dynamics and inter-chunk continuity. Experiments in simulation and on real robots demonstrate strong task performance with reduced inference latency. Real-world evaluations further show improved arm stability and gripper control compared with direct one-step sampling and DMD. These results support incorporating execution-relevant temporal structure into distillation for efficient robotic policies.

AI USE STATEMENT

In this work, generative AI tools were used to assist with Chinese–English translation, language editing, and identifying and correcting unclear or inconsistent descriptions in the manuscript. They were also used for limited code review and debugging, including identifying potential implementation errors and suggesting corrections. Generative AI tools were not used to formulate the research hypotheses, develop the proposed methodology, design the experiments, generate or modify experimental data, or analyze and interpret the experimental results. Tasks involving theoretical claims or proofs and qualitative data analysis were not applicable to this work. All AI-assisted text was manually reviewed and revised by the authors, and all AI-assisted code corrections were inspected and tested through the experiments reported in this paper. The authors take full responsibility for the final content of this work, including its text, claims, results, and software artifacts.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed experimental setup information in the appendix. Specifically, we report the source/checkpoint of the original teacher model, dataset configurations, training settings for various methods, and simulation evaluation settings. We will publicly release the code and data upon acceptance. We believe these details are sufficient to reproduce the main results.

REFERENCES

Arman Akbari, Ci Zhang, Arash Akbari, Lin Zhao, Yixiao Chen, Weiwei Chen, Xuan Zhang, Geng Yuan, and Yanzhi Wang. Flash-wam: Modality-aware distillation for world action models, 2026. *URL https://arxiv. org/abs/2606.05254*.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies, 2025. *URL https://arxiv. org/abs/2506.07339*, a.

Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking (2025). *URL https://arxiv. org/abs/2512.05964*, b.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

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

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.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $pi_0.5$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Bofang Jia, Pengxiang Ding, Can Cui, Mingyang Sun, Pengfang Qian, Siteng Huang, Zhaoxin Fan, and Donglin Wang. Score and distribution matching policy: Advanced accelerated visuomotor policies via matched distillation. *arXiv preprint arXiv:2412.09265*, 2024.

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

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

Yufeng Liu, Hang Yu, Juntu Zhao, Bocheng Li, Di Zhang, Mingzhu Li, Wenxuan Wu, Yingdong Hu, Junyuan Xie, Junliang Guo, et al. Learning native continuation for action chunking flow policies. *arXiv preprint arXiv:2602.12978*, 2026.

Wuyang Luan, Junhui Li, Weiguang Zhao, Wenjian Zhang, Tieru Wu, and Rui Ma. Snapflow: One-step action generation for flow-matching vlas via progressive self-distillation. *arXiv preprint arXiv:2604.05656*, 2026.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

Yifei Su, Ning Liu, Dong Chen, Zhen Zhao, Kun Wu, Meng Li, Zhiyuan Xu, Zhengping Che, and Jian Tang. Freqpolicy: Efficient flow-based visuomotor policy via frequency consistency. *Advances in Neural Information Processing Systems*, 38:27769–27797, 2026.

Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, et al. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024.

Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Hao Li, Hengtao Li, Jie Li, Jindi Lv, Jingyu Liu, et al. Gigaworld-policy: An efficient action-centered world–action model. *arXiv preprint arXiv:2603.17240*, 2026a.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026b.

Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024a.

Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6613–6623. IEEE, 2024b.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Yiming Zhong, Yumeng Liu, Chuyang Xiao, Zemin Yang, Youzhuo Wang, Yufei Zhu, Ye Shi, Yujing Sun, Xinge Zhu, and Yuexin Ma. Freqpolicy: Frequency autoregressive visuomotor policy with continuous tokens. *Advances in Neural Information Processing Systems*, 38:56493–56526, 2026.

A APPENDIX

This document supplements the main paper as follows:

- Sec. A.1 provides additional method details of OneRobo.
- Sec. A.2 describes the experimental settings, including simulation benchmarks, real-world robot setup, baselines and evaluation protocols.
- Sec. A.3 reports additional results on RoboTwin and real-world robotic tasks.

A.1 METHOD DETAILS

We provide the trajectory projection, distribution-matching objective, gripper supervision, and history construction used by OneRobo. An action chunk contains H time steps and D model-action channels.

A.1.1 TRAJECTORY-SUBSPACE PROPERTIES

Constructing the DCT basis. For a single-channel trajectory $x \in \mathbb{R}^H$, we use the fixed orthonormal DCT-II basis $B = [b_0, \dots, b_{H-1}]$. Its entries are

$$B_{t,k} = a_k \cos\left[\frac{\pi}{H}\left(t + \frac{1}{2}\right)k\right], \qquad a_k = \begin{cases} \sqrt{1/H}, & k = 0, \\ \sqrt{2/H}, & k > 0, \end{cases} \tag{13}$$

where $t, k \in \{0, \dots, H-1\}$. The columns are ordered by temporal frequency: b_0 is constant, and larger k represents faster variation. The basis depends only on H and is not learned.

Low-frequency projection. We retain $B_K = [b_0, \dots, b_{K-1}] \in \mathbb{R}^{H \times K}$, where $1 \le K \le H$. These columns span the trajectory subspace $\mathcal{S}_K = \{B_K q : q \in \mathbb{R}^K\}$. The associated projection is

$$P_K = B_K B_K^\top, \qquad P_K x = B_K(B_K^\top x). \tag{14}$$

Here $B_K^\top x$ extracts the retained DCT coefficients, and B_K reconstructs their contribution in the original time coordinates. Since $B_K^\top B_K = I_K$, we have $P_K^\top = P_K$ and $P_K^2 = B_K(B_K^\top B_K)B_K^\top = P_K$. Thus, projection maps any trajectory into $\mathcal{S}_K$ and leaves trajectories already in that subspace unchanged.

Applying the projection to action chunks. Let $\mathcal{J}_{\text{arm}}$ denote the arm channels. For $X \in \mathbb{R}^{H \times D}$, we define

$$[\mathcal{P}_K(X)]_{:,j} = \begin{cases} P_K X_{:,j}, & j \in \mathcal{J}_{\text{arm}}, \\ X_{:,j}, & j \notin \mathcal{J}_{\text{arm}}. \end{cases} \tag{15}$$

Consequently, each arm trajectory has K rather than H temporal degrees of freedom, while the action tensor remains $H \times D$. Gripper and other non-arm channels are unchanged by this operation.

Projected Gaussian noise. For one arm channel, sampling $z \sim \mathcal{N}(0, I_H)$ and projecting it gives

$$z_K = P_K z \sim \mathcal{N}(0, P_K). \tag{16}$$

Equivalently, the retained coefficients $q = B_K^\top z$ follow $\mathcal{N}(0, I_K)$, and $z_K = B_K q$. Thus, the noise is sampled in the K retained coefficient directions and reconstructed as an H-step trajectory in $\mathcal{S}_K$.

A.1.2 DISTRIBUTION-MATCHING OBJECTIVE

The frozen teacher T provides the reference velocity field. The student G generates one-step actions, and the trainable distribution estimator F_ϕ (Fake Score) learns the velocity field induced by the current student. All three receive the current condition $c = \mathcal{E}(o, \ell, s)$ from the frozen encoder. Only G and F_ϕ additionally receive history (p, w).

One-step student generation. Starting from Gaussian noise $z \sim \mathcal{N}(0, I)$, we construct the history-aware initialization $\tilde{z}$ as described in Appendix A.1.4. The student predicts one velocity and takes a single step from the noise endpoint to the action endpoint:

$$A_\theta = \mathcal{P}_K\left[\tilde{z} - \mathcal{P}_K\left[v_\theta^G(\tilde{z}, 1; c, p, w)\right]\right]. \tag{17}$$

The projections restrict the arm trajectories without modifying the native representation of the remaining channels.

Training the distribution estimator. To train F_ϕ, we detach the generated action and independently sample $\epsilon_{\mathrm{F}} \sim \mathcal{N}(0, I)$ and $\tau_{\mathrm{F}} \sim p_{\mathrm{F}}$. With $\epsilon_{K,\mathrm{F}} = \mathcal{P}_K(\epsilon_{\mathrm{F}})$, the noisy training sample is

$$x_{\mathrm{F}} = (1 - \tau_{\mathrm{F}})\,\mathrm{sg}(A_\theta) + \tau_{\mathrm{F}}\epsilon_{K,\mathrm{F}}, \tag{18}$$

where sg denotes stop-gradient. In this time convention, $\tau = 0$ corresponds to the action and $\tau = 1$ to noise. The derivative of the linear interpolation with respect to time gives the target velocity

$$v_{\mathrm{F}}^\star = \epsilon_{K,\mathrm{F}} - \mathrm{sg}(A_\theta). \tag{19}$$

The estimator is trained by

$$\mathcal{L}_F = \mathbb{E}\Big[\Big\langle \big(\mathcal{P}_K\left[v_\phi^F(x_{\mathrm{F}}, \tau_{\mathrm{F}}; c, p, w)\right] - v_{\mathrm{F}}^\star\big)^2 \Big\rangle_{\mathcal{M}}\Big]. \tag{20}$$

Here $\mathcal{M}$ is the action-validity mask, and $\langle R\rangle_{\mathcal{M}}$ denotes the sum of valid entries of R divided by their count, with a denominator of at least one. The mask is broadcast across the action channels before reduction. This loss updates only F_ϕ; its targets are generated by the student rather than taken from demonstration action labels.

Teacher and estimator predictions. For the generator update, we independently resample $\epsilon_{\mathrm{DM}} \sim \mathcal{N}(0, I)$ and $\tau_{\mathrm{DM}} \sim p_{\mathrm{DM}}$, and construct

$$x_{\mathrm{DM}} = (1 - \tau_{\mathrm{DM}})\,\mathrm{sg}(A_\theta) + \tau_{\mathrm{DM}}\mathcal{P}_K(\epsilon_{\mathrm{DM}}). \tag{21}$$

The estimator-training and distribution-matching branches need not use the same time distribution.

For a linear path, a velocity prediction v gives the clean-action estimate $x_\tau - \tau v$. We therefore query both networks at the same x_{DM} and τ_{DM}:

$$\begin{aligned} \hat{A}_T &= x_{\mathrm{DM}} - \tau_{\mathrm{DM}}\mathcal{P}_K[v_T(x_{\mathrm{DM}}, \tau_{\mathrm{DM}}; c)], \\ \hat{A}_F &= x_{\mathrm{DM}} - \tau_{\mathrm{DM}}\mathcal{P}_K\left[v_\phi^F(x_{\mathrm{DM}}, \tau_{\mathrm{DM}}; c, p, w)\right]. \end{aligned} \tag{22}$$

The teacher retains its original conditioning and does not receive student history.

Distribution-matching update. Following DMD Yin et al. (2024b;a), we use the difference between these estimates to define the student update direction. We first compute a scalar normalization for each sample:

$$d = \max\Big(\Big\langle |\,\mathrm{sg}(A_\theta) - \hat{A}_T| \Big\rangle_{\mathcal{M}}, \varepsilon_d\Big), \tag{23}$$

where the average is over that sample's valid action elements and $\varepsilon_d > 0$ prevents division by zero. The detached direction is

$$u_{\mathrm{DMD}} = \mathrm{sg}\left[\frac{\hat{A}_F - \hat{A}_T}{d}\right]. \tag{24}$$

To apply it without differentiating through either estimator, we construct a fixed regression target and minimize

$$\begin{aligned} A^\star &= \mathrm{sg}(A_\theta - u_{\mathrm{DMD}}), \\ \mathcal{L}_{\mathrm{DMD}} &= \frac{1}{2}\mathbb{E}\big[\langle (A_\theta - A^\star)^2\rangle_{\mathcal{M}}\big]. \end{aligned} \tag{25}$$

Because $A^\star$ is detached, differentiation yields the prescribed direction with the corresponding masking and averaging factors. Gradients reach the student through A_θ, not through the teacher, Fake Score, or noisy-state construction.

Subspace preservation. **Proposition.** If every arm column of $\tilde{z}$ lies in $\mathcal{S}_K$, the same holds for A_θ, x_{F}, x_{DM}, $\hat{A}_T$, $\hat{A}_F$, and u_{DMD} before loss masking.

Proof. The student output is projected by Eq. equation 17. Both endpoints of each interpolation are therefore in $\mathcal{S}_K$ for every arm channel, so their linear combination also belongs to it. Subtracting projected velocities preserves this property in Eq. equation 22. Finally, subtracting the two estimates and dividing by the scalar d preserves membership in the same subspace.

Joint objective and auxiliary weighting. The student also receives gripper supervision, defined in Appendix A.1.3:

$$\mathcal{L}_G = \mathcal{L}_{\mathrm{DMD}} + \alpha_{\mathrm{grip}}\mathcal{L}_{\mathrm{grip}}. \tag{26}$$

We set α_{grip} using the relative gradient magnitudes at the student action tensor, rather than the numerical magnitudes of the two losses. Specifically, over the batch, compute

$$n_D = \|\nabla_{A_\theta}\mathcal{L}_{\mathrm{DMD}}\|_2, \qquad n_g = \|\nabla_{A_\theta}\mathcal{L}_{\mathrm{grip}}\|_2. \tag{27}$$

The coefficient is then

$$\alpha_{\mathrm{grip}} = \begin{cases} \mathrm{sg}\left[\min\left(\alpha_{\max}, \beta_{\mathrm{aux}}\omega\dfrac{n_D}{\max(n_g, \varepsilon_g)}\right)\right], & n_g > \varepsilon_g, \\ 0, & \text{otherwise.} \end{cases} \tag{28}$$

Here β_{aux} specifies the auxiliary action-gradient budget, ω is its warmup multiplier, and $\alpha_{\max}$ caps the coefficient. The threshold ε_g handles negligible auxiliary gradients. Stop-gradient prevents differentiation through this weighting rule.

The estimator and student use separate optimizers, with the update ratio and numerical settings reported in Appendix A.2.4. Only the student and frozen condition encoder are required at deployment.

A.1.3 Gripper Temporal Supervision

Gripper channels are not DCT-truncated. Instead, their command levels and temporal changes are matched to a frozen multi-step teacher, allowing teacher transitions to be retained rather than smoothed away.

Teacher reference in control coordinates. We obtain $A_T^{\mathrm{aux}} = \mathrm{sg}[T^{(10)}(z; c)]$ using the same raw Gaussian draw z as the student. The teacher uses its original ten-step sampler, without history mixing or velocity projection. Both action chunks are then decoded with the same fixed, differentiable action codec:

$$\begin{aligned} g_G &= \mathrm{Grip}[\mathcal{C}_{\mathrm{dec}}(A_\theta; s)], \\ g_T &= \mathrm{Grip}[\mathcal{C}_{\mathrm{dec}}(A_T^{\mathrm{aux}}; s)]. \end{aligned} \tag{29}$$

Here $g_G, g_T \in \mathbb{R}^{H\times d_g}$ contain d_g gripper-command channels in controller coordinates. They represent issued commands, not measured physical gripper states.

Comparing three temporal orders. For a gripper trajectory g, define the discrete differences

$$\begin{aligned} (D_0 g)_t &= g_t, && 0 \le t < H, \\ (D_1 g)_t &= g_t - g_{t-1}, && 1 \le t < H, \\ (D_2 g)_t &= g_t - 2g_{t-1} + g_{t-2}, && 2 \le t < H. \end{aligned} \tag{30}$$

The discrete time step is one. The three orders compare command levels, adjacent changes, and changes of those changes, respectively. Each is matched to the teacher; the objective does not penalize the student's temporal differences toward zero.

Valid frames and temporal weights. Let $\nu_t \in \{0, 1\}$ indicate a valid action position. A first-order difference requires two valid frames, and a second-order difference requires three. We additionally assign frame weight $a_t = 1$ for $t < E$ and $a_t = \beta_{\mathrm{tail}}$ for $t \ge E$. For order $r \in \{0, 1, 2\}$, the resulting weight is

$$\Omega_{t,j}^{(r)} = \left(\prod_{k=0}^{r} \nu_{t-k}\right) \min_{0\le k\le r} a_{t-k}, \qquad r \le t < H. \tag{31}$$

The product excludes differences involving invalid frames; the minimum assigns the lowest weight among their contributing frames. These weights are shared across the d_g gripper channels.

Robust temporal loss. We measure each residual with the Huber penalty

$$\rho_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta, \\ \delta(|e| - \frac{1}{2}\delta), & |e| > \delta. \end{cases} \tag{32}$$

For each temporal order, compute its own weighted average:

$$\mathcal{L}_r = \frac{\sum_{t=r}^{H-1} \sum_{j=1}^{d_g} \Omega_{t,j}^{(r)} \rho_{\delta_r}((D_r g_G - D_r g_T)_{t,j})}{\max\left(\sum_{t=r}^{H-1} \sum_{j=1}^{d_g} \Omega_{t,j}^{(r)}, 1\right)}. \tag{33}$$

Batch indices are omitted; the training reduction also includes the batch dimension. We combine the three terms as

$$\mathcal{L}_{\text{grip}} = \mathcal{L}_0 + 0.5\mathcal{L}_1 + 0.25\mathcal{L}_2. \tag{34}$$

Huber thresholds and the tail weight are specified with the training configuration. The teacher reference and loss are used only during training, and no learned gripper decoder is added.

A.1.4 History Prior Construction

History supplies the current generator with the unexecuted part of the previous plan. We first align that plan, identify its usable positions, and then provide it through both noise initialization and action-token conditioning.

Aligning the previous plan. Let A^- be the previous action chunk, generated under state s^-. After its first E actions have been executed, where $1 \le E < H$, the current state is s. We decode the old plan, retain its unexecuted tail, and re-encode it under the current state:

$$\begin{aligned} U^- &= \mathcal{C}_{\text{dec}}(A^-; s^-), \\ \bar{p} &= \text{Pad}_H\left[\mathcal{C}_{\text{enc}}(U^-_{E:H}; s)\right]. \end{aligned} \tag{35}$$

Here $U^-_{E:H}$ contains the remaining $H - E$ commands, and Pad_H appends zeros after re-encoding to produce H model-action positions. For state-relative joint representations, this procedure preserves the old absolute targets under the new reference state. For policies such as LIBERO, the codec instead preserves the native relative-command semantics.

Selecting valid history. Let ν_t and ν_t^- be the current and previous validity masks, extended by zero outside the horizon. The aligned mask is $m_t = \nu_t \nu^-_{E+t}$, so a historical action is usable only when both corresponding positions are valid. We assign it the weight

$$w_{t,j} = b\, m_t\, \mathbb{I}[j \in \mathcal{J}_{\text{arm}}]\, \max(1 - t/O, 0), \tag{36}$$

where b is one when history is retained and zero otherwise. The positive decay horizon O determines the direct support of the prior, whereas E determines how far the previous plan is shifted. Gripper and padded channels receive zero history weights.

Let $S = \mathbb{I}[w > 0]$ be the binary support mask. The prior is

$$p = S \odot \mathcal{P}_K(S \odot \bar{p}), \tag{37}$$

where $\odot$ denotes element-wise multiplication. The first mask removes unsupported entries before projection; the second removes values spread outside the support by the DCT. This final masking means that p need not itself lie in the low-frequency subspace.

History-conditioned noise initialization. With $z_K = \mathcal{P}_K(z)$, active history modifies the initial noise as

$$\tilde{z} = \mathcal{P}_K[(1 - \lambda_{\text{hist}} w) \odot z_K + \lambda_{\text{hist}} w \odot p]\,. \tag{38}$$

The coefficient λ_{hist} controls the mixing strength, while w determines its variation across time and channels. The outer projection returns the arm initialization to $\mathcal{S}_K$ even when the masked prior is not in that subspace. When history is disabled, we use $\tilde{z} = z_K$ directly. A limited input support does not imply a strictly local output effect: temporal projection can propagate the influence beyond the support.

Additive action-token conditioning. The prior and its weights are also supplied explicitly to both trainable networks. If h^i_{act} denotes the original action-token representation, we use

$$\widetilde{h}^i_{\text{act}} = h^i_{\text{act}} + pW^i_p + wW^i_w, \qquad i \in \{G, F\}. \tag{39}$$

The two matrices have shape $D \times d_i$, where d_i is the action-token width. They separately encode the historical action values and their weights. Each network has its own bias-free, zero-initialized projections; the frozen prefix and teacher do not receive these embeddings.

Training-time history. We pair frame n with frame $n - E$ from the same episode and instruction, not with an adjacent minibatch entry. For eligible pairs, history is retained with $b \sim \text{Bernoulli}(0.9)$. When retained, the student generates a detached previous plan without history conditioning:

$$A^- = \text{sg}[G_\theta(z^-; c^-, 0, 0)], \qquad z^- \sim \mathcal{N}(0, I). \tag{40}$$

This plan is converted to (p, w) using the construction above. Dropped or invalid history gives $p = w = 0$. Only one previous chunk is bootstrapped; the current observation remains the recorded dataset observation, not the result of executing that generated plan.

Deployment. At deployment, the previous plan is cached from the preceding replanning cycle, so no additional bootstrap evaluation is required. History is cleared at resets or invalid alignment. Unlike the single-bootstrap training procedure, deployment reuses recursively generated plans. The two history distributions are therefore not assumed to be identical.

A.2 Experimental Details

A.2.1 Common Protocol and Backbone Configurations

We distill pretrained teacher policies into one-step action generators, rather than train complete policies from scratch. The original teachers and visual-language conditioning components remain frozen. Training updates the action-generation networks, their associated projections, and method-specific trainable modules. Table 6 summarizes the datasets, model inputs, action horizons, and training configurations.

An *outer step* denotes one iteration of a method's training loop, not necessarily one student optimizer update. Within each benchmark, methods share the effective batch size and nominal outer-step budget. Student update frequencies, auxiliary-model updates, and teacher sampling costs remain method-specific.

A.2.2 LIBERO Experiments

Data and preprocessing. Both $\pi_{0.5}$ and SmolVLA use the same local LIBERO dataset, comprising 1,693 trajectories and 273,465 frames across 40 tasks. Each observation contains two camera views and an eight-dimensional proprioceptive state. The environment accepts seven-dimensional actions, consisting of six arm channels and one gripper channel, while both backbones use a 32-dimensional internal action representation. The model-side image resolution is 224×224 for $\pi_{0.5}$. SmolVLA resizes and pads images to 512×512 within the model.

Training budget and optimization. All compared methods use an effective batch size of 64, distributed as 32 examples per GPU on two A100 GPUs, and run for 2,000 outer steps. Trainable action parameters are maintained in FP32. The training seed is 6666, and the global gradient norm

Table 6: Dataset, input, and training configurations. H is the prediction horizon, and E is the number of actions executed before replanning. Image sizes refer to model-side inputs; the RoboTwin input is a packed three-camera representation. The nominal RoboTwin budget is 5,000 outer steps.

Setting	$\pi_{0.5}$[a]	SmolVLA[b]	FastWAM[c]
Benchmark	LIBERO	LIBERO	RoboTwin
Teacher checkpoint	Checkpoint[d]	Checkpoint[e]	Checkpoint[f]
Training tasks	40	40	50
Training trajectories	1,693	1,693	2,500
Camera views	2	2	3
Model-side image size	224×224	512×512	384×320
Environment action dimension	7	7	14
Proprioceptive state dimension	8	8	14
Prediction horizon H	10	50	32
Execution horizon E	5	1	24
Effective batch size	64	64	128
Training devices	2 A100 GPUs	2 A100 GPUs	4 A100 GPUs
Nominal outer steps	2,000	2,000	5,000

[a] $\pi_{0.5}$ code: `https://github.com/Physical-Intelligence/openpi`
[b] SmolVLA code: `https://github.com/huggingface/lerobot/tree/8515d456be1dbef8c133f07188c785e683eca899/src/lerobot/policies/smolvla`
[c] FastWAM code: `https://github.com/yuantianyuan01/FastWAM`
[d] $\pi_{0.5}$ teacher checkpoint: `gs://openpi-assets/checkpoints/pi05_libero`
[e] SmolVLA teacher checkpoint: `https://huggingface.co/HuggingFaceVLA/smolvla_libero`
[f] FastWAM teacher checkpoint: `https://huggingface.co/yuanty/fastwam`

Table 7: Optimization settings on LIBERO. Student update counts correspond to 2,000 outer steps. LR denotes learning rate. The auxiliary column gives the distribution-estimator learning rate where applicable. EMA denotes an exponential moving-average model without an independent optimizer; a dash indicates no auxiliary model.

Method	Optimizer	Student LR $\pi_{0.5}$	Student LR SmolVLA	Auxiliary LR / EMA	Student updates
DMD2	AdamW	5×10^{-7}	5×10^{-7}	2.5×10^{-6}	400
OneRobo	AdamW	5×10^{-7}	5×10^{-7}	2.5×10^{-6}	400
OneDP	Adam	1×10^{-6}	1×10^{-6}	2×10^{-5}	2,000
FreqPolicy	AdamW	2.5×10^{-5}	1×10^{-6}	–	2,000
OFP	AdamW	2.5×10^{-5}	2.5×10^{-5}	EMA	2,000

is clipped to 1.0. Learning rates are held constant during distillation; schedules from the original teacher training configurations are not inherited.

Table 7 reports the method-specific learning rates and student update counts. DMD2 and OneRobo updates their distribution estimator at every outer step and their student every fifth outer step, resulting in 2,000 estimator updates and 400 student updates. The other methods perform 2,000 student updates. For FreqPolicy on SmolVLA, the student learning rate is tuned to 1×10^{-6}.

OneRobo uses AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and weight decay 10^{-10}. FreqPolicy and OFP also use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay 10^{-10}. OneDP uses Adam with $(\beta_1, \beta_2) = (0, 0.999)$.

A.2.3 RoboTwin Experiments

Data and teacher initialization. The RoboTwin dataset contains 2,500 training trajectories and 500 separate validation trajectories across 50 tasks, with 50 training trajectories per task.

Table 8: Optimization settings on RoboTwin. r_G is the number of outer steps between student updates. Counts refer to completed 5,000-outer-step runs. Auxiliary-column notation follows Table 7.

Method	Optimizer	Student LR	Auxiliary LR / EMA	r_G	Student updates
DMD2	AdamW	2.5×10^{-6}	2.5×10^{-5}	5	1,000
OneRobo	AdamW	2.5×10^{-6}	2.5×10^{-5}	5	1,000
OneDP	Adam	1×10^{-6}	2×10^{-5}	1	5,000
FreqPolicy	AdamW	2.5×10^{-5}	–	1	5,000

Inputs and trainable components. Each condition consists of current images from three cameras, a language instruction, and a 14-dimensional proprioceptive state. The FastWAM predicts 32 actions with 14 dimensions per action. Each camera image is resized to 240×320; the multi-view packing pipeline then produces a 384×320 model input. Although the dataset configuration includes video windows, only the current observation is encoded for distillation. Future images are not used as conditioning inputs.

The original teacher and video-conditioning branch remain frozen, and language embeddings are cached. Training updates the ActionDiT and method-specific modules. The ActionDiT contains 30 layers, with hidden width 1,024 and feed-forward width 4,096.

Optimization and update schedules. All methods use four NVIDIA A100 80GB GPUs, an effective batch size of 128, and global gradient clipping at 1.0. The nominal budget is 5,000 outer steps. The run configurations named `Raw-DMD` and `Structured` are reported as DMD2 and OneRobo, respectively. Both update the estimator every outer step and the student every fifth outer step. A completed run therefore contains 5,000 estimator updates and 1,000 student updates.

Table 8 reports the method-specific optimization settings. DMD2 and OneRobo use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and zero weight decay. OneDP uses Adam with $(\beta_1, \beta_2) = (0, 0.999)$. FreqPolicy use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay 10^{-10}.

Microbatching and precision. All configurations use 32 examples per GPU without gradient accumulation and maintain an effective batch size of 128. OneDP and FreqPolicy maintain FP32 action parameters and optimizer states while using BF16 autocast. The Raw-DMD run uses BF16 action parameters.

Baseline adaptations. In the adapted OneDP implementation, the distribution estimator is updated before the student, training time is sampled uniformly from $[0.02, 0.95]$, and the Raw-DMD denominator normalization is not used.

The adapted FreqPolicy objective combines a flow loss with full-spectrum DCT consistency losses on velocities and endpoints, including gripper channels. This differs from OneRobo's truncated arm-only trajectory projection. This implementation uses neither a distribution estimator nor an EMA model.

A.2.4 ONEROBO-SPECIFIC SETTINGS

Table 9 lists the principal OneRobo hyperparameters. Temporal DCT truncation applies only to arm channels. The table separates the retained mode count K from the history support/decay horizon O and from the execution horizon E in Table 6.

LIBERO auxiliary supervision. Teacher targets use ten sampling steps, and the distribution-matching time range is $[0.05, 0.95]$. The gripper Huber threshold is 0.05 for action values and both temporal difference orders. Gripper supervision is warmed up over the first 100 student updates, and the unexecuted tail receives relative weight 0.25. Gripper outputs retain their native representation.

RoboTwin distribution matching. DMD2 and OneRobo share the same underlying distribution-matching settings. Estimator training samples time from $\mathrm{Beta}(1.5, 1.0)$ with a configured minimum

Table 9: Principal OneRobo hyperparameters. K is the number of retained temporal DCT modes, and O is the history support/decay horizon. The gripper weights correspond to command values and first- and second-order temporal differences.

Setting	$\pi_{0.5}$	SmolVLA	FastWAM
Retained DCT modes K	3	16	16
Number of projected arm channels	6	6	12
History support horizon O	3	8	8
History retention probability	0.9	0.9	0.9
History prior strength λ_{hist}	0.5	0.5	0.5
Gripper value weight λ_0	1.0	1.0	1.0
Gripper first-difference weight λ_1	0.5	0.5	0.5
Gripper second-difference weight λ_2	0.25	0.25	0.25
Gripper action-gradient budget	0.05	0.05	0.05

noise level of 0.001. Distribution matching uses time values in $[0.05, 0.95]$ and a denominator stabilization constant of 10^{-4}.

A.2.5 Evaluation Settings

All evaluated student policies use one action-network function evaluation per predicted chunk (NFE = 1). Within each backbone, methods share the prediction and execution horizons in Table 6: $(H, E) = (10, 5)$ for $\pi_{0.5}$, $(50, 1)$ for SmolVLA, and $(32, 24)$ for RoboTwin. All simulation evaluations and inference latency measurements on LIBERO and RoboTwin are conducted using a single NVIDIA A100 GPU per task. For real-world experiments, policy inference is performed on a single NVIDIA RTX 4090 GPU.

For LIBERO, we evaluate all four task suites, each containing 10 tasks. For π_{05}, each task is evaluated for 50 episodes per seed, and the complete evaluation is repeated using three different global evaluation seeds. For SmolVLA, we use two different seeds. For RoboTwin, we evaluate all 50 tasks under both the Easy and Hard settings, with 50 episodes per task in each setting. This yields 2,500 episodes per setting and 5,000 episodes in total per evaluated method. For each backbone and simulation setting, all compared methods use the same evaluation seeds to maintain a consistent randomization protocol across methods. These evaluation seeds correspond to repeated evaluations of the same trained checkpoint, rather than independent training runs.

A.2.6 Real-world Robot Settings

We conduct real-world experiments on an ALOHA-AgileX bimanual robot using a $\pi_{0.5}$-based fine-tuned policy. The policy is deployed in a server-client manner: the inference server loads the policy checkpoint, tokenizer, and normalization statistics, and performs model inference on GPU, while the robot-side client collects observations through ROS and sends them to the server for action prediction. Each inference input consists of three RGB views, including one front camera and two wrist cameras, together with the 14-dimensional joint state of the two robot arms. The robot publishes joint commands at 30 Hz with a default action chunk size of 50.

A.3 Additional Results

A.3.1 RoboTwin Full Experiments Results.

Following the training and inference configurations described above, Tab. 10 and Tab. 11 present comparisons of the success rates of various methods under the easy and hard settings, respectively. Under the easy setting, our OneRobo method drops by only 1.64% compared with the original teacher model, whereas the best-performing among the other tested methods drops by 3.64%. Under the hard setting, our method still maintains its advantage, dropping by only 2.24% compared with the teacher model.

Table 10: Success rates (%) on 50 RoboTwin tasks under the `demo_clean` setting.

Task	FastWAM 10-step	FastWAM 1-step	DMD2 1-step	OneDP 1-step	FreqPolicy 1-step	OneRobo 1-step
adjust_bottle	100	98	100	100	94	100
beat_block_hammer	100	78	96	100	92	98
blocks_ranking_rgb	100	92	98	100	92	100
blocks_ranking_size	94	74	94	92	92	92
click_alarmclock	100	96	100	100	100	100
click_bell	100	100	100	100	98	100
dump_bin_bigbin	98	90	94	96	84	90
grab_roller	100	100	100	100	100	100
handover_block	98	40	86	84	2	96
handover_mic	98	48	72	74	98	96
hanging_mug	70	48	54	68	34	72
lift_pot	100	20	92	100	56	100
move_can_pot	88	48	86	76	70	84
move_pillbottle_pad	100	92	100	96	92	94
move_playingcard_away	100	98	98	98	96	100
move_stapler_pad	84	26	58	74	20	84
open_laptop	96	94	90	88	92	96
open_microwave	74	8	20	70	2	64
pick_diverse_bottles	86	80	86	82	88	84
pick_dual_bottles	100	84	94	96	98	98
place_a2b_left	96	92	98	100	96	100
place_a2b_right	92	84	92	90	92	90
place_bread_basket	100	90	92	94	94	92
place_bread_skillet	94	80	92	90	82	88
place_burger_fries	94	82	92	94	98	92
place_can_basket	74	56	64	74	78	76
place_cans_plasticbox	100	44	100	100	92	100
place_container_plate	98	92	98	100	98	100
place_dual_shoes	96	70	80	96	46	90
place_empty_cup	100	98	100	100	100	100
place_fan	98	58	82	96	66	92
place_mouse_pad	86	56	60	74	82	84
place_object_basket	80	78	78	82	86	82
place_object_scale	100	76	84	94	86	92
place_object_stand	94	80	86	88	92	92
place_phone_stand	88	92	92	92	72	90
place_shoe	88	86	88	84	84	86
press_stapler	92	90	88	92	90	94
put_bottles_dustbin	92	16	24	72	66	64
put_object_cabinet	86	54	78	82	56	88
rotate_qrcode	78	82	92	94	56	92
scan_object	94	48	80	88	62	88
shake_bottle	100	96	100	100	100	100
shake_bottle_horizontally	100	96	100	100	100	100
stack_blocks_three	96	62	92	90	76	96
stack_blocks_two	100	82	100	100	98	100
stack_bowls_three	88	60	78	78	64	90
stack_bowls_two	96	88	94	96	96	96
stamp_seal	94	42	58	62	58	82
turn_switch	60	60	74	62	40	74
Avg. (50 tasks)	**92.80**	72.08	85.08	89.16	78.12	**91.16**

A.3.2 Real World Task

Fig. 7 and Fig. 8 compare different methods on the *Pick and Place the Fork* and *Fold the Clothes* tasks. OneRobo can perform these two tasks well under 1-step inference. In contrast, both the teacher model with direct 1-step inference and DMD with 1-step inference exhibit action jitter.

Table 11: Success rates (%) on 50 RoboTwin tasks under the `demo_randomized` setting.

Task	FastWAM 10-step	FastWAM 1-step	DMD2 1-step	OneDP 1-step	FreqPolicy 1-step	OneRobo 1-step
adjust_bottle	100	98	100	100	100	100
beat_block_hammer	98	76	100	98	94	96
blocks_ranking_rgb	98	86	98	98	86	98
blocks_ranking_size	100	78	90	94	80	96
click_alarmclock	100	100	100	100	100	100
click_bell	100	100	100	100	100	100
dump_bin_bigbin	100	90	88	92	92	92
grab_roller	100	100	100	100	100	100
handover_block	86	28	74	70	10	76
handover_mic	98	68	88	88	92	94
hanging_mug	62	48	54	68	24	60
lift_pot	98	30	88	96	86	98
move_can_pot	98	70	86	82	76	88
move_pillbottle_pad	98	84	100	100	90	100
move_playingcard_away	98	96	98	94	92	96
move_stapler_pad	74	26	78	74	12	66
open_laptop	100	100	100	100	94	100
open_microwave	70	8	2	36	32	56
pick_diverse_bottles	84	76	88	88	88	82
pick_dual_bottles	96	88	94	98	96	96
place_a2b_left	100	92	96	100	100	96
place_a2b_right	94	86	92	94	98	94
place_bread_basket	96	60	84	90	84	98
place_bread_skillet	94	78	90	92	88	90
place_burger_fries	98	78	94	100	98	94
place_can_basket	66	52	60	68	74	62
place_cans_plasticbox	98	70	96	100	96	100
place_container_plate	94	98	92	98	100	96
place_dual_shoes	84	70	82	74	48	82
place_empty_cup	98	98	100	98	100	100
place_fan	94	50	92	88	66	96
place_mouse_pad	88	52	76	80	78	88
place_object_basket	80	56	78	74	76	80
place_object_scale	94	76	84	84	88	96
place_object_stand	98	76	96	96	92	98
place_phone_stand	96	98	92	96	90	98
place_shoe	90	88	92	94	90	94
press_stapler	90	90	92	90	90	90
put_bottles_dustbin	86	14	32	62	40	60
put_object_cabinet	92	68	80	76	72	86
rotate_qrcode	86	76	88	76	56	88
scan_object	88	34	74	82	58	80
shake_bottle	100	98	100	100	98	100
shake_bottle_horizontally	100	98	100	100	100	100
stack_blocks_three	96	46	86	94	60	94
stack_blocks_two	100	88	98	100	100	100
stack_bowls_three	94	56	66	78	68	82
stack_bowls_two	90	96	94	98	96	100
stamp_seal	90	44	72	60	64	88
turn_switch	66	58	64	48	50	62
Avg. (50 tasks)	**91.96**	71.88	85.36	87.32	79.24	**89.72**

In the pick-and-place fork task, this manifests as failure to grasp the fork or failure to place it. In fold the clothes, unstable grasping leads to repeating the same action for an extended period. This highlights that OneRobo can reduce inference latency on real-robot tasks while maintaining performance comparable to the teacher model.

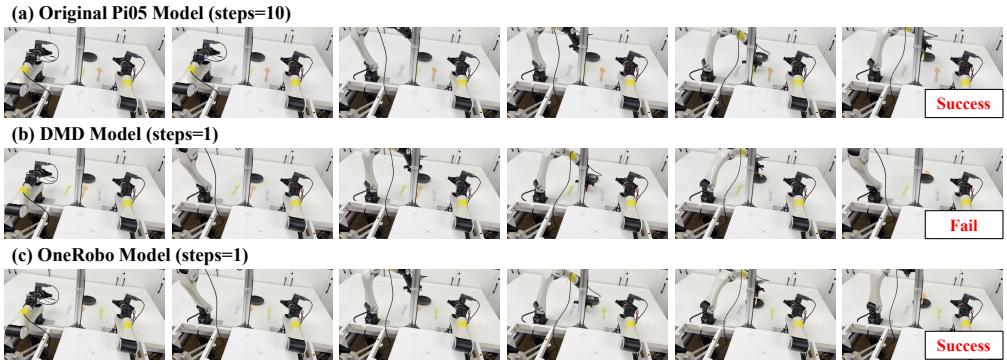


Figure 7: Comparison on the real-world *Pick and Place the Fork* task.

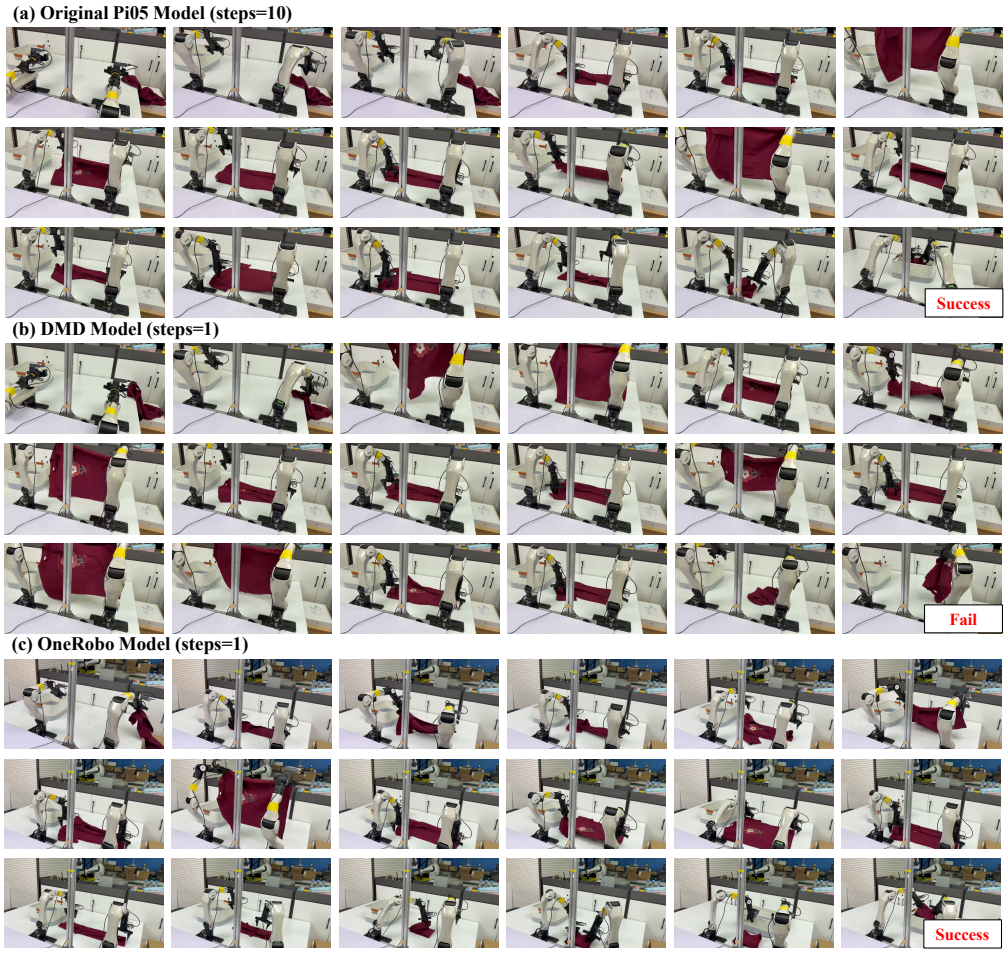


Figure 8: Qualitative comparison on the real-world *Fold_the_Clothes* task.

A.3.3 Projection Placement and Channel Ablation

We examine projection placement and channel selection using trained SmolVLA policies on LIBERO-Spatial. RTC is disabled in all configurations, and each configuration is evaluated over 500 episodes. We report task success, translational and rotational direction-reversal rates, and first-order gripper temporal-difference error.

Table 12: Projection placement and channel ablation on LIBERO-Spatial with SmolVLA. Each configuration is evaluated over 500 episodes with RTC disabled. Bold indicates the best observed value in each column.

Configuration	Success rate (%) ↑	Translation reversal (%) ↓	Rotation reversal (%) ↓	Gripper D_1 error ↓
No DCT	78.60	1.467	0.720	0.1433
Inference-only DCT	76.00	1.461	0.748	0.1441
Output smoothing	78.00	1.420	0.783	0.1451
Noise + output projection	78.20	**1.301**	**0.564**	0.1398
Projected DMD (arm-only)	**79.80**	1.363	0.693	**0.1258**
Projected DMD (all-channel)	78.00	1.515	0.712	0.1417

Projection placement. As shown in Table 12, arm-only projected DMD achieves the highest observed success rate of 79.80%, exceeding the no-DCT and inference-only DCT variants by 1.20 and 3.80 percentage points, respectively. It also obtains the lowest gripper temporal-difference error. Noise and output projection yields the lowest translational and rotational reversal rates, but achieves only 78.20% success. Thus, lower reversal rates alone do not imply higher task success.

Channel selection. Extending projected DMD from arm channels to all action channels reduces success from 79.80% to 78.00% and increases gripper temporal-difference error from 0.1258 to 0.1417. These observations favor channel-specific projection over uniform projection in this setting.