ONLINE ROBOT POLICY REFINEMENT VIA DYNAMICS–VALUE DECOMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Pretrained imitation policies can solve complex manipulation tasks, yet small errors can still cause failure in precise manipulation. Online reinforcement learning (RL) can correct such failures, but most policy refinement methods rely on Q-functions whose offline initialization can be unreliable for the nearby unseen actions introduced during refinement. Rather than directly pretraining a Q-function, we compose a pretrained latent dynamics model, adapted with task-specific offline transitions, with a state-value model initialized through task-progress supervision. We call this *dynamics–value decomposition*: an action chunk is evaluated by predicting its latent successor and scoring the predicted state. This composition provides an initial action evaluator that generalizes locally to action perturbations during refinement. During online RL, new transitions update the dynamics model, TD learning calibrates the value model to environment returns, and policy gradients propagate through their composition. Instantiated with DINO-WM and RoboMeter, our action evaluator replaces direct Q-functions in residual and latent-steering policy refinement methods. Across precise manipulation tasks in simulation and the real world, our method improves online sample efficiency over the corresponding Q-based methods.

1 INTRODUCTION

Imitation learning has produced capable robot policies for complex manipulation tasks from expert demonstrations (Dasari et al., 2024; Physical Intelligence et al., 2025). Yet, small errors in grasping, alignment, or contact can still cause failures, particularly in precise manipulation. Online reinforcement learning (RL) can correct such failures through additional interaction (Nakamoto et al., 2023; Luo et al., 2024; 2025), and recent methods refine pretrained policies through residual actions, latent steering, or parameter-efficient adaptation (Sun & Song, 2026; Dong et al., 2026a; Yuan et al., 2025; Wagenmaker et al., 2025; Im et al., 2026). Most of these methods rely on a learned Q-function to decide how the base policy should be modified.

A central challenge is obtaining a useful critic before substantial online interaction. A Q-function must assign long-horizon return to each state–action pair, coupling an action's immediate consequence with the continuation policy and sparse task rewards. Offline critic pretraining can provide an initialization, but its estimates are mostly reliable on the state–action distribution represented in the offline data (Nakamoto et al., 2023; Dodeja et al., 2026; Dong et al., 2026b). During policy refinement, the policy needs to explore nearby but unseen corrections without direct return labels, requiring the critic to extrapolate both how these actions change the state and whether the resulting state supports task completion.

Our key observation is that these two questions admit complementary offline supervision. We start from a pretrained action-conditioned latent dynamics model and adapt it using offline transitions to predict where an action leads, while a state-value model can be initialized from task-progress predictions to estimate how desirable the resulting state is. Their composition can therefore evaluate a nearby candidate action by predicting its latent successor and scoring that state. This form of local compositional generalization is well matched to policy refinement, where candidate actions remain close to those of a competent base policy.

We propose *dynamics–value decomposition*, which uses this composition as an action evaluator for online robot policy refinement. We initialize the dynamics model with a pretrained DINO-WM

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.

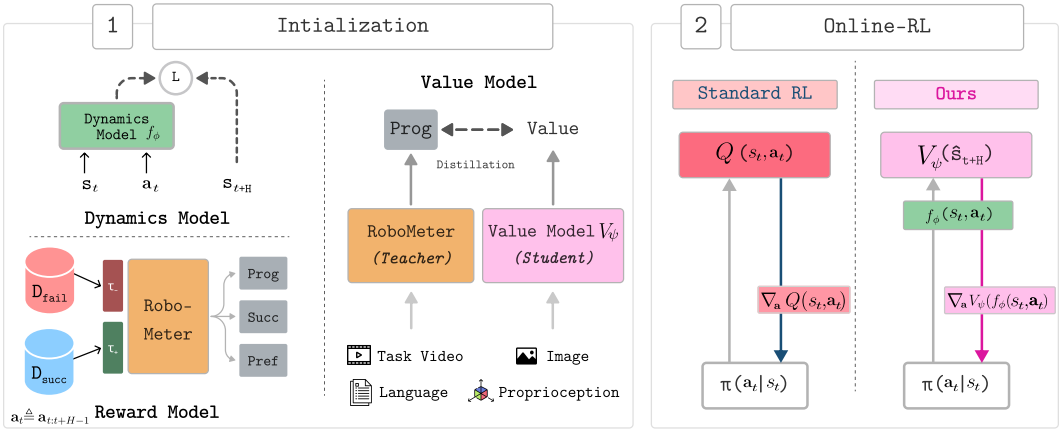


Figure 1: **Component-wise initialization for online policy refinement.** A direct Q-critic has no reliable target for a nearby action not observed in the offline data. Our dynamics-value decomposition separately (i) adapts a pretrained action-conditioned latent dynamics model using offline transitions and (ii) initializes a task-informed state-value model from failure-aware progress supervision. Their composition scores counterfactual action chunks before substantial online interaction and is subsequently calibrated using online transitions and sparse environment rewards.

model (Zhou et al., 2025) and adapt it with task-specific offline transitions, while initializing the state-value model with failure-aware task-progress predictions from RoboMeter (Liang et al., 2026). During online RL, newly collected transitions update the dynamics model, TD learning calibrates the value model to environment returns, and the policy is optimized by backpropagating through their composition.

We use a single action-chunk prediction to focus on local corrections while avoiding error accumulation over long imagined rollouts. The resulting action evaluator can replace a direct Q-critic in both residual-action and latent-steering refinement methods. Our approach is related to model-based value gradient methods (Heess et al., 2015; Byravan et al., 2020; Hafner et al., 2020), but its focus is component-wise initialization of an action evaluator for post-training strong robot policies rather than the value-gradient identity itself.

Our contributions are threefold:

- We identify a critic-initialization bottleneck in online robot policy refinement: an offline-trained direct Q-function can be unreliable for the nearby but unseen actions introduced during refinement.
- We introduce dynamics–value decomposition, which composes pretrained latent dynamics with a state-value model initialized from failure-aware task-progress supervision to evaluate candidate action chunks.
- We integrate the proposed evaluator into residual-action and latent-steering refinement methods and show improved online sample efficiency on precise manipulation tasks in simulation and the real world.

2 Related Work

Online refinement of pretrained robot policies. Online reinforcement learning (RL) can improve imitation learning policies through additional interaction without learning the task from scratch (Nakamoto et al., 2023; Luo et al., 2024; 2025). Recent methods preserve a competent behavior prior by restricting adaptation to lightweight interfaces such as residual actions or policy latents (Yuan et al., 2025; Ankile et al., 2025; Dong et al., 2026a; Im et al., 2026). In particular, DICE-RL (Sun & Song, 2026) freezes a pretrained diffusion- or flow-based policy and learns an action-space residual with off-policy RL; its action-chunk Q-critic guides residual updates, selective behavior regularization, and value-guided selection among candidate chunks. DSRL (Wagenmaker

et al., 2025) instead freezes a diffusion policy and trains an actor in its latent-noise space, using an actor–critic objective to steer the decoded action distribution without modifying the base-policy parameters. We use DICE-RL and DSRL as complementary residual-action and latent-steering instantiations: rather than changing their policy parameterizations or interaction procedures, we replace their direct Q-based action evaluation with our dynamics–value decomposition.

Offline-to-online critic initialization. Offline-to-online RL commonly initializes a policy and a value function from prior data, but the subsequent critic can be sensitive to the offline behavior distribution, its initial value targets, and the policy shift induced by online improvement (Nakamoto et al., 2023; Dodeja et al., 2026; Dong et al., 2026b). This issue is particularly relevant for policy refinement, where the online actor deliberately proposes nearby actions that are absent from the offline trajectories and therefore lack direct return labels. Prior work primarily studies how to better initialize, calibrate, or transfer a direct Q-function across this transition. Our method takes a complementary approach: it avoids relying on a monolithic offline Q initialization by separately adapting pretrained latent dynamics using transitions and initializing state values from task-progress supervision, then calibrating the composed evaluator with online TD learning.

Model-based value estimation. Model-based RL uses learned dynamics to generate synthetic experience, improve value targets, plan actions, or differentiate policy objectives. MBPO trains a model-free learner on short model rollouts (Janner et al., 2019), while model-based value expansion incorporates predicted transitions into value targets (Feinberg et al., 2018). Stochastic and imagined value-gradient methods backpropagate future values through differentiable dynamics (Heess et al., 2015; Byravan et al., 2020), and Dreamer optimizes policies through longer latent imagined trajectories (Hafner et al., 2020). DINO-WM further shows that action-conditioned dynamics can be learned from offline robot trajectories in a pretrained visual feature space and used for planning (Zhou et al., 2025). In contrast, we neither generate synthetic replay nor rely on long-horizon imagination or test-time planning; we use a single action-chunk prediction to construct an offline-initialized action evaluator for local corrections around a strong pretrained policy.

Pretrained reward and value models. Recent robot-learning systems combine predictive models with progress or value estimators to improve policies from predicted outcomes. RISE (Yang et al., 2026), for example, generates visual futures with a world model, evaluates them using a progress value model, and trains a policy from imagined advantages. Dense robot reward models likewise provide language-guided, stage-aware, process-level, or trajectory-comparison supervision beyond sparse task rewards (Zhang et al., 2025; Chen et al., 2026; Tan et al., 2026; Liang et al., 2026). Our method shares the idea of separating outcome prediction from task evaluation, but uses the composition differently: RoboMeter supplies an initialization for a state-value model rather than replacing the environment objective, online TD learning calibrates that model to actual returns, and the one-step dynamics–value score serves as a differentiable critic inside real online RL rather than as supervision from multi-step imagined rollouts.

3 Approach

Our goal is to improve a pretrained policy with minimal additional environment interaction. Instead of initializing a direct Q-function from sparse and binary rollout returns, we construct an action evaluator by composing two complementary priors: a pretrained latent dynamics model adapted to task-specific transitions (Section 3.3) and a state-value model initialized through task-progress supervision (Section 3.4). The resulting evaluator scores a candidate action by predicting its latent successor state and evaluating the predicted state. During online RL, the dynamics and value models are updated with newly collected rollouts, and the policy is optimized through their composition (Section 3.5). Figure 1 summarizes the framework.

3.1 Problem Setup

We consider a manipulation task modeled as an episodic MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$. The state $s_t = (o_t, p_t) \in \mathcal{S}$ contains multi-view observations o_t and robot proprioception p_t. Each low-level action $a_t \in \mathcal{A}$ specifies either a 6D end-effector target or a 7D joint-position target, together with a 1D gripper command. The environment provides a sparse binary task reward r_t, where r_t is one when

the task first succeeds and zero otherwise. An episode terminates upon success or at a fixed time limit. The objective is to maximize the expected discounted return $J(\pi) = \mathbb{E}_\pi[\sum_{t\geq 0} \gamma^t r_t]$, which corresponds to maximizing task success while favoring earlier completion when $\gamma < 1$.

We assume access to a pretrained policy $\pi_{\text{prior}}(\mathbf{a}_t \mid s_t)$, where $\mathbf{a}_t = (a_t, a_{t+1}, ..., a_{t+H-1})$ denotes a chunk of H low-level actions. Our framework is agnostic to how the base policy was trained and does not require access to its original training data $\mathcal{D}_{\text{demo}}$. In our experiments, π_{prior} is flow-based action-chunking policy trained on $\mathcal{D}_{\text{demo}}$ following Dasari et al. (2024).

Before online RL, we roll out the base policy and retain both successful and failed trajectories, denoted by $\mathcal{D}_+$ and $\mathcal{D}_-$. Together, they form the offline dataset $\mathcal{D}_{\text{off}} = \mathcal{D}_+ \cup \mathcal{D}_-$. Both subsets provide transition supervision for adapting the dynamics model, while their contrasting task outcomes provide task-progress supervision for initializing the value model. Our goal is to improve π_{prior} using $\mathcal{D}_{\text{off}}$ and a limited budget of additional online interaction $\mathcal{D}_{\text{on}}$.

3.2 Dynamics–Value Decomposition

We first describe the action-evaluation problem at the action-chunk resolution. Let $R_t^{(H)} = \sum_{k=0}^{H-1} \gamma^k r_{t+k}$ denote the reward accumulated over one chunk, and let $d_t^{(H)}$ indicate whether the episode terminates during the chunk. Because the reward is sparse and terminal, $R_t^{(H)}$ is zero unless task success occurs within the chunk. For a continuation policy π, the corresponding action value is

$$Q_H^\pi(s_t, \mathbf{a}_t) = \mathbb{E}\left[R_t^{(H)} + (1 - d_t^{(H)})\gamma^H V^\pi(s_{t+H}) \mid s_t, \mathbf{a}_t\right]. \tag{1}$$

A direct Q-function must jointly estimate how the action chunk changes the state and how valuable the resulting state is under the continuation policy. Offline rollout data provide return supervision only for action chunks that were actually executed, whereas online policy refinement introduces nearby but previously unseen corrections. A direct critic must therefore extrapolate both the local consequence and the long-horizon value of each candidate action.

We separate these two prediction problems. Let $z_t = e(s_t) = [\varphi(o_t); p_t]$ denote a latent state representation, where φ is a frozen DINOv2 visual encoder (Oquab et al., 2024). An action-conditioned latent dynamics model predicts the successor representation after one action chunk,

$$\hat{z}_{t+H} = f_\psi(z_t, \mathbf{a}_t), \tag{2}$$

and a state-value model V_ϕ evaluates the predicted successor. Their composition defines the *dynamics–value action score*

$$S_{\text{DV}}(s_t, \mathbf{a}_t) = V_\phi(f_\psi(e(s_t), \mathbf{a}_t)). \tag{3}$$

Unlike a monolithic Q-function, the two components admit complementary supervision: every rollout transition provides a target for the dynamics model, while sparse TD targets and task-progress labels supervise the value model.

The composition is intended for local counterfactual action evaluation. Given a base-policy action $\tilde{\mathbf{a}}$ and a nearby correction $\boldsymbol{\delta}$, the dynamics model predicts the consequence of $\tilde{\mathbf{a}} + \boldsymbol{\delta}$, and the value model evaluates the predicted state, rather than directly extrapolating a return for the unseen state–action pair. The method therefore targets the modest action shifts induced while refining a competent base policy, rather than arbitrary out-of-distribution actions.

The score in Eq. (3) approximates the successor-value term in Eq. (1), rather than the complete action value. A complete model-based approximation would additionally predict the reward and termination associated with the action chunk,

$$\widehat{Q}_H(s_t, \mathbf{a}_t) = \widehat{R}_t^{(H)}(s_t, \mathbf{a}_t) + \left(1 - \widehat{d}_t^{(H)}(s_t, \mathbf{a}_t)\right) \gamma^H S_{\text{DV}}(s_t, \mathbf{a}_t). \tag{4}$$

In our sparse-reward tasks, every non-terminal chunk has $R_t^{(H)} = 0$ and $d_t^{(H)} = 0$. Because such chunks dominate the local policy updates before task completion, we use S_{DV} as the differentiable action-evaluation signal without treating $V_\phi \circ f_\psi$ as an exact Q-function.

Because both components are differentiable, the action derivative follows from the chain rule:

$$\frac{\partial S_{\text{DV}}(s, \mathbf{a})}{\partial \mathbf{a}} = \frac{\partial V_\phi(\hat{z})}{\partial \hat{z}} \frac{\partial f_\psi(z, \mathbf{a})}{\partial \mathbf{a}}, \qquad z = e(s), \quad \hat{z} = f_\psi(z, \mathbf{a}). \tag{5}$$

The dynamics derivative captures how an action correction changes the predicted successor representation, while the value derivative identifies changes associated with greater task value.

3.3 ONE-STEP LATENT DYNAMICS MODEL

Initialization. We instantiate the latent dynamics model f_ψ using the pretrained DINO-WM (Zhou et al., 2025), which operates in a frozen DINOv2 feature space (Oquab et al., 2024), and adapt its future prediction head to the target robot, task, and action-chunk distribution using only the base-policy rollouts in $\mathcal{D}_{\text{off}}$ (i.e. we only finetune the prediction head, not its backbone). We adopt DINO-WM because its pretrained visual representation provides a structured feature space, allowing the dynamics model to focus on learning action-conditioned state transitions rather than learning visual representation from scratch. For a transition $(s_t, \mathbf{a}_t, s_{t+H})$, we minimize

$$\mathcal{L}_{\text{dyn}}(\psi; \mathcal{B}) = \mathbb{E}_{(s_t, \mathbf{a}_t, s_{t+H}) \sim \mathcal{B}} \left[\| f_\psi(e(s_t), \mathbf{a}_t) - e(s_{t+H}) \|_2^2 \right]. \tag{6}$$

Using $\mathcal{B} = \mathcal{D}_{\text{off}}$ yields the task-adapted parameters ψ_0. This objective requires neither task rewards nor return labels, so both successful and failed rollouts provide dense supervision. We predict a single action chunk rather than a long imagined rollout, focusing the model on the local consequences needed for policy refinement while avoiding compounding prediction error.

Online refinement. During online RL, rollouts from the evolving policy are added to $\mathcal{D}_{\text{on}}$, and we define $\mathcal{D}_{\text{replay}} = \mathcal{D}_{\text{off}} \cup \mathcal{D}_{\text{on}}$. Starting from ψ_0, we continue updating the dynamics model by minimizing $\mathcal{L}_{\text{dyn}}(\psi; \mathcal{D}_{\text{replay}})$. These updates adapt the model to the action distribution induced by the improving policy while retaining coverage of the base-policy rollouts used for initialization. The visual encoder remains frozen throughout, and only the future state prediction head is updated.

3.4 STATE-VALUE MODEL

Initialization. The value model operates on the same latent state representation $z_t = e(s_t) = [\varphi(o_t); p_t]$. For each camera view, a learned multi-query attention readout aggregates the frozen DINOv2 patch embeddings before the resulting visual features are concatenated with proprioception p_t. We use an ensemble of M value networks,

$$V_\phi(z_t) = \frac{1}{M} \sum_{m=1}^{M} V_{\phi_m}(z_t). \tag{7}$$

Before online refinement, we initialize the ensemble solely through task-progress distillation; no TD updates are performed at this stage. RoboMeter (Liang et al., 2026) assigns each frame in a completed trajectory a progress target $\hat{\rho}_t \in [0, 1]$ conditioned on the task instruction. Let $\mathcal{D}_{\text{demo}}^{\text{prog}}$ denote an optional set of expert demonstrations used only for progress distillation, and define $\mathcal{D}_{\text{off}}^{\text{prog}} = \mathcal{D}_{\text{off}} \cup \mathcal{D}_{\text{demo}}^{\text{prog}}$. We set $\mathcal{D}_{\text{demo}}^{\text{prog}} = \varnothing$ in simulation. In the real-world experiments, we additionally include expert demonstrations in this progress-supervision set.

For each transition $(s_t, \mathbf{a}_t, s_{t+H})$, we supervise both the observed successor latent z_{t+H} and its dynamics-predicted counterpart $\hat{z}_{t+H} = f_\psi(z_t, \mathbf{a}_t)$ using the progress target of the actual successor:

$$\mathcal{L}_{\text{prog}}(\phi; \mathcal{B}) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{(s_t, \mathbf{a}_t, s_{t+H}, \hat{\rho}_{t+H}) \sim \mathcal{B}} \left[\ell(V_{\phi_m}(z_{t+H}), \hat{\rho}_{t+H}) + \ell(V_{\phi_m}(\hat{z}_{t+H}), \hat{\rho}_{t+H}) \right], \tag{8}$$

where ℓ is the MSE loss. During initialization, we set $\psi = \psi_0$ and hold the dynamics model fixed while minimizing this objective. Applying the same target to the observed and predicted successors exposes the value model to the representations it will receive during policy refinement and encourages consistency between the two pathways. We obtain the initial value parameters as

$$\phi_0 = \arg\min_\phi \mathcal{L}_{\text{prog}}(\phi; \mathcal{D}_{\text{off}}^{\text{prog}}). \tag{9}$$

Before online RL, V_{ϕ_0} can be interpreted as a task-progress-based state scorer rather than a return-calibrated value function. Thus, S_{DV} initially provides a dense ranking of nearby candidate actions.

Online refinement. TD learning begins only after online refinement starts. For a chunk-level transition $(s_t, \mathbf{a}_t, R_t^{(H)}, s_{t+H}, d_t^{(H)})$, we define

$$y_t^{\text{TD}} = R_t^{(H)} + (1 - d_t^{(H)}) \gamma^H V_{\bar{\phi}}(z_{t+H}), \tag{10}$$

where $V_{\bar{\phi}}$ is a Polyak-averaged target ensemble. For a transition dataset $\mathcal{B}$, the TD objective is

$$\mathcal{L}_{\mathrm{TD}}(\phi;\mathcal{B}) = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathcal{B}}\left[\ell\big(V_{\phi_m}(z_t), y_t^{\mathrm{TD}}\big)\right], \tag{11}$$

where ℓ is the Smooth L1 loss. Starting from ϕ_0, we apply TD updates on $\mathcal{D}_{\mathrm{replay}}$ to calibrate the value model toward the expected return and adapt it to the state distribution induced by the evolving policy. Task-progress supervision provides the initial dense structure, while online TD learning grounds V_ϕ in the environment and trains S_{DV} to approximate the successor-value term in Eq. (1).

3.5 Online Policy Refinement

To apply the evaluator to different policy-refinement algorithms, let $g_\theta(s,\xi)$ denote the differentiable mapping from state s and policy noise ξ to an action chunk, including any frozen base-policy components. We replace the direct-Q actor term of the underlying algorithm with

$$\mathcal{L}_{\mathrm{actor}}(\theta) = -\mathbb{E}_{s,\xi}\left[\gamma^H S_{\mathrm{DV}}(s, g_\theta(s,\xi))\right] + \mathcal{R}_{\mathrm{alg}}(\theta), \tag{12}$$

where $\mathcal{R}_{\mathrm{alg}}$ contains the entropy, behavior-regularization, or other policy terms used by the underlying algorithm. During a policy update, the dynamics and value parameters are held fixed while gradients propagate through their composition to θ.

For **residual-action refinement**, the frozen base policy first proposes $\tilde{\mathbf{a}} \sim \pi_{\mathrm{prior}}(\cdot \mid s)$, and a residual policy predicts a bounded correction,

$$g_\theta(s,\xi) = \tilde{\mathbf{a}} + \alpha\,\boldsymbol{\delta}_\theta(s,\tilde{\mathbf{a}},\xi), \tag{13}$$

where α controls the correction magnitude and keeps candidate actions within the local region covered by the dynamics model. For **latent steering**, g_θ includes the frozen base-policy decoder, while the trainable policy modifies its latent or noise input; S_{DV} evaluates the resulting decoded action chunk. Whenever the original algorithm uses a direct Q-function for policy optimization or candidate-action selection, we substitute the dynamics–value score while retaining the remaining interaction procedure and algorithm-specific regularization. The overall training loop alternates online rollout collection, dynamics refinement, value refinement, and policy improvement.

4 Experiments

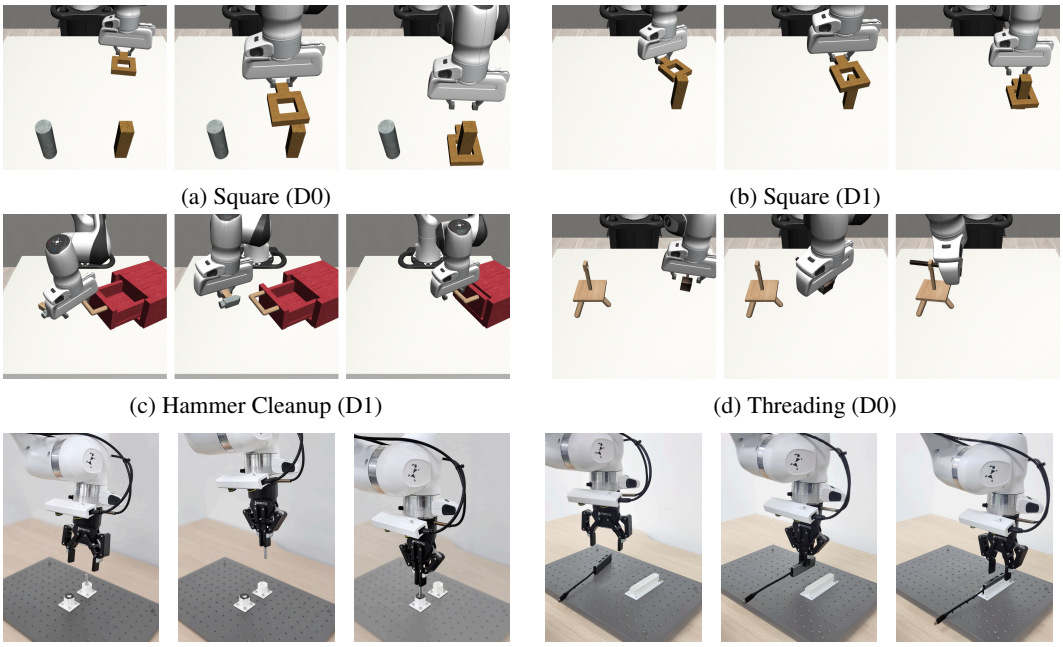
We evaluate whether the proposed dynamics–value decomposition enables more sample-efficient online refinement of pretrained robot policies. Our experiments focus on precise manipulation tasks in which the base policy often makes substantial task progress but can still fail because of small errors in grasping, alignment, or contact. We address three questions: (i) does the proposed action evaluator improve online sample efficiency over direct Q-function learning, (ii) are its benefits consistent across different policy refinement approaches, and (iii) how does the quality of the adapted latent dynamics model affect downstream policy improvement?

4.1 Comparison Methods

We compare direct Q-function learning with the proposed dynamics–value decomposition within two representative online policy refinement algorithms:

- **DICE-RL**: DICE-RL (Sun & Song, 2026) performs action-space refinement by learning a residual correction to the action chunk proposed by a frozen base policy. In the original method, a directly learned action-chunk Q-function guides residual-policy optimization and, where applicable, value-based selection among candidate actions.
- **DSRL**: DSRL (Wagenmaker et al., 2025) performs latent-space refinement by learning to steer the latent or noise input of a frozen generative base policy. The selected latent is decoded into an action chunk, and a learned critic guides the steering policy toward actions with higher predicted value.

For each refinement algorithm, we compare the following action evaluators:



(a) Square (D0) (b) Square (D1)

(c) Hammer Cleanup (D1) (d) Threading (D0)

(e) Peg Insertion (f) USB Hub Insertion

Figure 2: **Illustration of the six evaluation settings.** The top two rows show four simulated settings across three tasks: Square (D0 and D1), Hammer Cleanup (D1), and Threading (D0). The bottom row shows the two real-world insertion tasks (Peg Insertion and USB Hub Insertion).

- **Direct-Q**: the original formulation of each refinement algorithm: an action-chunk critic for DICE-RL and a latent critic for DSRL.
- **Dynamics–Value (Ours)**: the direct Q-function score is replaced by $V_\phi(f_\psi(e(s_t), \mathbf{a}_t))$, while keeping the remaining online RL procedure unchanged.

Within each refinement framework, the two variants use the same base policy, offline rollout data, online interaction budget, replay procedure, action parameterization, and algorithm-specific regularization. The comparison therefore isolates the effect of replacing direct action-value estimation with dynamics–value decomposition.

4.2 Tasks and Evaluation Protocol

We evaluate four simulated precise manipulation settings from MimicGen (Mandlekar et al., 2023): Square D0, Square D1, Hammer Cleanup D1, and Threading D0. We additionally evaluate two real-world tasks: Peg Insertion and USB Hub Insertion. These tasks require accurate grasping and spatial alignment, followed by sustained contact during operations such as insertion, threading, and hammering. Small local errors can therefore cause failure even after the base policy has completed most of the task.

For each task, we train a flow-based DiT action-chunking policy (Dasari et al., 2024) from expert demonstrations. We use 1,000 demonstrations per simulated task and 52 and 50 demonstrations for the real-world Peg Insertion and USB Hub Insertion tasks, respectively. We then collect successful and failed rollouts from the base policy to construct $\mathcal{D}_{\text{off}}$. These rollouts are used to adapt the latent dynamics model and initialize the state-value model. All compared methods begin from the same base policy and offline rollout dataset.

We measure online sample efficiency using task success rate as a function of environment interaction. In simulation, each policy checkpoint is evaluated over 100 episodes, and we report the mean and standard deviation over three random seeds. For the real-world experiments, we report the trailing 20-episode moving average of online success. In all figures, the dotted horizontal line denotes the success rate of the initial base policy.

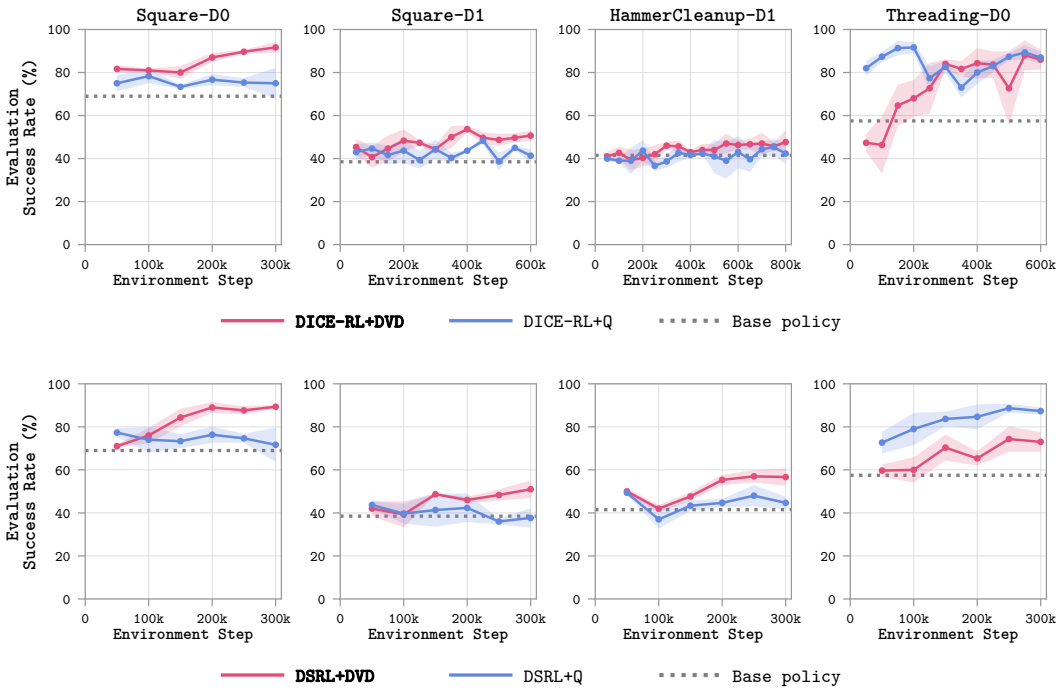


Figure 3: **Success rate during online policy refinement in simulation.** We compare dynamics–value decomposition with direct-Q refinement with DICE-RL (top) and DSRL (bottom). Each checkpoint is evaluated over 100 trials. Shaded regions show one standard deviation over three seeds; the dotted line indicates the base policy.

4.3 Results

Simulation results. Figure 3 compares the two action evaluators under the same online interaction budget. Across Square D0, Square D1, and Hammer Cleanup D1, dynamics–value decomposition improves refinement under both DICE-RL and DSRL. Averaged over all training checkpoints, it achieves a 6.4 percentage-point higher success rate across the three tasks and both refinement methods, with the gap increasing to 12.4 percentage points at the final checkpoint. This improvement is observed under both latent steering and residual policy refinement, suggesting that the benefit stems from the value signal rather than a particular refinement mechanism.

Threading D0 is the main exception: dynamics–value decomposition does not outperform direct-Q refinement on this setting. When Threading D0 is included, the average improvement over all checkpoints and all eight task–algorithm combinations decreases to 1.5 percentage points, although the average improvement at the final checkpoint remains 7.4 percentage points. We hypothesize that Threading requires particularly small geometric differences between the thread and its target. Because both the dynamics and value models operate on DINOv2 features, the shared latent representation may not preserve the fine-grained visual distinctions needed to evaluate these corrections accurately. This result highlights an important limitation of the approach: its effectiveness depends on the latent representation retaining the task-relevant details required for precise local action evaluation.

Real-world results. We next evaluate whether the proposed action evaluator transfers to physical manipulation, where interaction is more costly and observations, contacts, and robot execution are less controlled than in simulation. Figure 4 reports online refinement on Peg Insertion and USB Hub Insertion under DICE-RL. Both our method (DICE-RL+DVD) and direct Q-based refinement (DICE-RL+Q) improve upon the base policy, but their relative performance differs across tasks. On Peg Insertion, online performance is highly variable and the two methods are often comparable, although DICE-RL+DVD reaches higher success rates at several stages and ends above DICE-RL+Q.

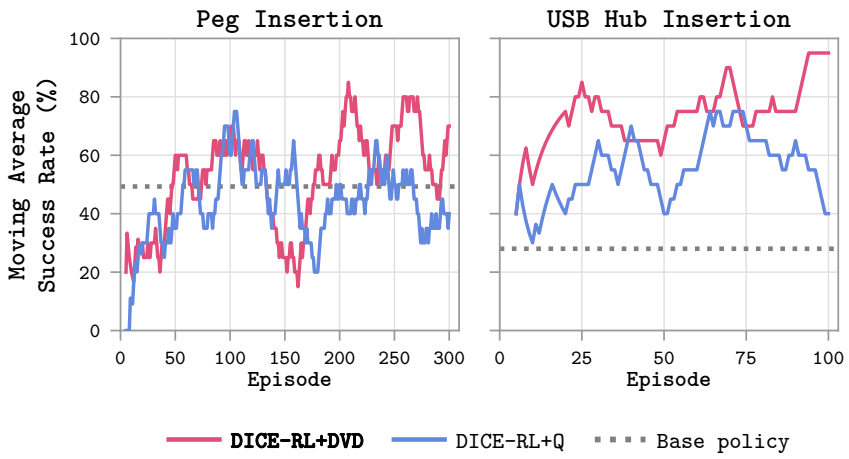


Figure 4: **Success rate during online policy refinement on real-world tasks.** We compare our dynamics–value estimator against direct Q-function-based policy refinement, applied to DICE-RL. Curves show the trailing 20-episode moving average of the success rate for each run. The dotted line indicates the success rate of the base policy.

On USB Hub Insertion, DICE-RL+DVD rapidly surpasses the base policy, generally remains above DICE-RL+Q over the shared 100-episode budget, and reaches approximately 95% success at the final checkpoint. DICE-RL+Q exhibits larger fluctuations and achieves lower success under the same interaction budget.

Overall, these results provide real-world evidence consistent with the simulation experiments, while also illustrating that the magnitude of the benefit can vary substantially across tasks. Due to the limited real-world experimental time, we evaluate only DICE-RL in the physical experiments.

Effect of dynamics model training. We study whether longer dynamics model training improves downstream policy learning by varying the number of DINO-WM training steps. As shown in Figure 5, increasing the training duration consistently improves the cumulative success rate. While all variants make comparable progress early in training, their performance gradually diverges, with the model trained for 300K steps achieving the highest final success rate, followed by 150K and 50K steps. These results suggest that further optimizing the dynamics model provides a more accurate learning signal for online RL, leading to improved downstream policy performance.

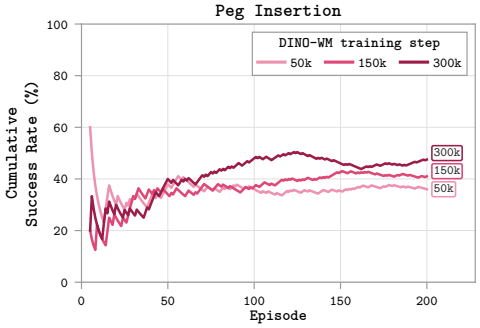


Figure 5: **Dynamics model training step ablation.**

5 Conclusion

We introduced dynamics–value decomposition for online refinement of pretrained robot policies. The method builds on the observation that direct Q-functions lack reliable return labels for the nearby counterfactual actions introduced by a refined policy, whereas a pretrained action-conditioned latent dynamics model can be adapted with task-specific transitions and a state-value model can be initialized from state-centric task-progress supervision. Their composition provides an initial action evaluator for local policy corrections; online transition learning and TD updates then calibrate it to the evolving policy and environment return. Across precise manipulation tasks, this factorized evaluator improves the sample efficiency of multiple online policy refinement mechanisms relative to direct-Q counterparts.

AI USE STATEMENT

In this work, we used generative AI tools for proofreading of the manuscript, including checking grammar, wording, and readability, and for final cleanup and organization of the code for open-source release. We have not used generative AI tools to generate synthetic datasets, help develop theoretical models or conceptual frameworks, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, and interpret results. All AI-assisted text and code were reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide our source code and instructions for reproducing the reported results in the supplementary material.

REFERENCES

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual off-policy rl for finetuning behavior cloning policies. *arXiv preprint arXiv:2509.19301*, 2025.

Arunkumar Byravan, Jost Tobias Springenberg, Abbas Abdolmaleki, Roland Hafner, Michael Neunert, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Imagined value gradients: Model-based policy optimization with transferable latent dynamics models. In *Conference on Robot Learning*, pp. 566–589. PMLR, 2020.

Qianzhong Chen, Justin Yu, Mac Schwager, Pieter Abbeel, Fred Shentu, and Philipp Wu. Sarm: Stage-aware reward modeling for long horizon robot manipulation. In *International Conference on Learning Representations*, 2026.

Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

Lakshita Dodeja, Ondrej Biza, Shivam Vats, Stephen Hart, Stefanie Tellex, Robin Walters, Karl Schmeckpeper, and Thomas Weng. When life gives you bc, make q-functions: Extracting q-values from behavior cloning for on-robot reinforcement learning. In *Robotics: Science and Systems*, 2026.

Perry Dong, Kuo-Han Hung, Tian Gao, Dorsa Sadigh, and Chelsea Finn. Expo-ft: Sample-efficient reinforcement learning finetuning for vision-language-action models. In *Conference on Robot Learning*, 2026a.

Perry Dong, Ron Polonsky, Dorsa Sadigh, and Chelsea Finn. Do you really need to pretrain q-functions for online rl fine-tuning? *arXiv preprint arXiv:2607.27203*, 2026b.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. In *International Conference on Machine Learning*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, volume 28, pp. 2944–2952, 2015.

Hokyun Im, Andrey Kolobov, Jianlong Fu, and Youngwoon Lee. Latent policy steering through one-step flow policies. In *Robotics: Science and Systems*, 2026.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Neural Information Processing Systems*, volume 32, 2019.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems*, 2024.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Biyik, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *Robotics: Science and Systems*, 2026.

Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pp. 16961–16969. IEEE, 2024.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, pp. 1820–1864. PMLR, 2023.

Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. In *Neural Information Processing Systems*, volume 36, pp. 62244–62269, 2023.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research Journal*, pp. 1–31, 2024.

Physical Intelligence, Kevin Black, Noah Brown, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Conference on Robot Learning*, 2025.

Zhanyi Sun and Shuran Song. From prior to pro: Efficient skill mastery via distribution contractive rl finetuning. In *International Conference on Machine Learning*, 2026.

Huajie Tan, Sixiang Chen, Yijie Xu, Zixiao Wang, Yuheng Ji, Cheng Chi, Yaoxu Lyu, Zhongxia Zhao, Xiansheng Chen, Peterson Co, et al. Robo-dopamine: General process reward modeling for high-precision robotic manipulation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2026.

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

Jiazhi Yang, Kunyang Lin, Jinwei Li, Wencong Zhang, Tianwei Lin, Longyan Wu, Zhizhong Su, Hao Zhao, Ya-Qin Zhang, Li Chen, et al. Rise: Self-improving robot policy with compositional world model. In *Robotics: Science and Systems*, 2026.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. In *International Conference on Learning Representations*, 2025.

Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. RewiND: Language-guided rewards teach robot policies without new demonstrations. In *Conference on Robot Learning*, 2025.

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *International Conference on Machine Learning*, pp. 79115–79135, 2025.

A Additional Experimental Details

A.1 Hardware Setup

We use a hardware setup based on DROID (Khazatsky et al., 2024). The system consists of a Franka Research 3 Facelift robot arm equipped with a Robotiq 2F-85 gripper. Visual observations are captured from two viewpoints: a side-view ZED 2i camera and a wrist-mounted ZED Mini camera. Both cameras record RGB images at a resolution of 720×1280 and a frame rate of 10 Hz.

A.2 Dynamics Model Architecture

Our dynamics model follows DINO-WM (Zhou et al., 2025): a 6-layer Transformer predictor operating on frozen DINOv2 ViT-S/14 patch features (14×14 patches per camera). Given the patch features of two context frames ($t-8$ and t) from both cameras, the 8-step action chunk $a_{t:t+8}$ and the proprioceptive state s_t, it predicts the patch features of frame $t+8$ in a single forward pass. We depart from DINO-WM in how the prediction is formed. DINO-WM concatenates action and proprioception embeddings to every patch token, trains a frame-causal predictor with teacher forcing, and reads each frame's next-frame prediction from that frame's own output tokens. We instead encode the action chunk (with a 1-D convolution over its steps) and the state as one token each, and append one learnable query token per target patch position and camera (2×196). The predictor attends bidirectionally over the context, action, state and query tokens, and the query outputs are the predicted features of frame $t+8$. Because only the context frames are inputs and the target frame is not, no causal mask is needed. A small MLP head on the state token's output predicts the proprioceptive change $s_{t+8} - s_t$, so the value function receives a predicted state alongside the predicted features. Both outputs are trained with a mean-squared-error loss.

A.3 Value Model Architecture

The value function V_ϕ reads the same frozen DINOv2 ViT-S/14 patch features as the dynamics model, so it scores observed and predicted frames alike. A multi-query attention readout compresses the 2×196 patch tokens: each camera has $K = 4$ learnable queries $q_{c,k} \in \mathbb{R}^{384}$. Each query returns the softmax-weighted mean of the raw patch features, weighted by $\mathrm{softmax}_i\big(\langle z_{c,i}, q_{c,k}\rangle / \sqrt{384}\big)$, with no key or value projections. The 8 pooled vectors are concatenated, linearly projected to 768-D, and concatenated with the 9-D proprioceptive state. Three MLP heads (3×256, LayerNorm, Mish) share this readout, and V_ϕ is their mean. Because V_ϕ is back-propagated through at every actor update, and with DICE-RL also ranks 16 candidate chunks per decision, this readout keeps it cheap: 8×196 dot products per frame instead of self-attention over all 392 tokens. V_ϕ is first fit offline by MSE to RoboMeter progress, on observed and predicted features of base-policy rollouts. It is then trained online by TD on the ensemble mean, with a Polyak target ($\tau = 0.005$).

A.4 Real-world Task Setup

Table 1: Episode length on the real-world tasks, in control steps. Maximum is the episode time limit. The mean is taken over all episodes collected during online training by DICE-RL+DVD and DICE-RL+Q (600 episodes for peg insertion, 302 for USB hub insertion)

Task	Mean	Maximum
Peg Insertion	233	400
USB Hub Insertion	171	400

A.5 HYPERPARAMETERS

Table 2: Hyperparameters for online RL in the action space (residual refinement; MimicGen, image observations). Both methods act on 8-step action chunks of the same frozen flow-matching prior. Where the methods differ, entries read *Q-based Refinement / Ours (DVD)*; †Ours (DVD) only.

Hyperparameter	**Value**
Environment	
Parallel environments	1
Action chunk	8 steps × 7-D; executed action $= \mathrm{clip}(\tilde{\mathbf{a}} + \boldsymbol{\delta}, -1, 1)$
Prior integration steps	10
Discount γ per primitive step (γ^8)	0.99 (0.923)
Reward	sparse: 1 on the chunk in which the task succeeds
Time-limit truncation	terminal, no bootstrap (horizon 400; hammer 500)
Optimisation	
Actor objective	min over 10 Q heads / $\gamma^8 V_\phi(f_\psi(s, \mathbf{a}))$
Actor learning rate	1×10^{-4}
Critic (value) learning rate	1×10^{-4}
World-model learning rate†	1×10^{-4}
Batch size (transitions)	256 / 64
Update frequency	every chunk / every 4 chunks[a]
Gradient steps per update	1 critic + 1 actor / 1 value + 1 actor + 1 world model
Policy update frequency	every update (no delay)
Target update rate τ	0.01 / 0.005
Critic (value) ensemble	10 heads, min / 3 nets, mean
Gradient clipping (actor, critic / value)	10
Residual regularisation	$100 \cdot \frac{1}{56}\|\boldsymbol{\delta}\|_2^2$ (batch mean); residual scale 1, no norm cap
Value initialisation	random / RoboMeter progress (offline)
Online progress distillation†	every 20 updates[b]
Data	
Replay buffer (chunks)	100,000
Demonstration : online sampling	$1.0 \rightarrow 0.1$ over the first 40k env steps, then 0.1
Warm-up	none
Exploration	learned policy w.p. $\min(1, t/75\text{k})$, otherwise a prior draw
Candidate selection	best of 16 prior draws, ranked by $\min_i Q_i$ / by $\gamma^8 V_\phi \circ f_\psi$
Exploration noise	none (deterministic residual)
Total env steps	see note[c]
Networks	
Cameras; resolution	agentview + wrist; 224×224
Actor encoder	IMPALA, 128 per camera: the critic's (detached) / its own
DrQ random shift	4 px on actor, critic and target inputs / 4 px on actor inputs only
Critic (value) input	IMPALA shared by the 10 heads / frozen DINOv2-S/14 + per-camera 4-query attention readout
World model†	DINO-WM (6 layers, 6 heads, width 384)
MLPs	3×256, LayerNorm, Mish
Proprioception	9-D

[a] Hammer: every chunk. [b] Square_d0 only; the other tasks use TD only. [c] Square_d0 / square_d1 / threading / hammer: Q-based Refinement 300k / 600k / 600k / 600k; Ours (DVD) 300k / 600k / 300k / 600k.

Table 3: Hyperparameters for online RL in the noise space of the frozen prior (latent steering). Both methods learn a policy $\pi_W(w \mid s)$ over the prior's initial noise and differ mainly in the signal the actor ascends. Where the methods differ, entries read *Q-based Refinement / Ours (DVD)*; †Ours (DVD) only; ‡Q-based Refinement only.

Hyperparameter	**Value**
Environment	
Parallel environments	1
Action chunk; prior integration steps	8 steps × 7-D; 10
Noise dimension	56 (8 executed rows × 7); remaining rows resampled
Noise box	$[-1, 1]$
Discount γ per primitive step (γ^8)	0.99 (0.923)
Reward	sparse: 1 on the chunk in which the task succeeds
Time-limit truncation	terminal, no bootstrap (horizon 400; hammer 500)
Optimisation	
Actor objective	min over noise critics Q^W / $\gamma^8 V_\phi(f_\psi(s, \pi_{\text{prior}}(s, w)))$
Actor learning rate	1×10^{-4}
Critic (value) learning rate	1×10^{-4}
World-model learning rate†	1×10^{-4}
Temperature learning rate; initial α	1.5×10^{-3}; 1.0
Target entropy	0 (summed over the 56 noise dimensions)
Actor $\log \sigma$ lower bound; activation	-5; Mish
Batch size (transitions)	256 / 64
Update frequency	every chunk / every 4 chunks
Gradient steps per update	1 critic, actor and α every 2nd update / 2 actor + 1 value + 1 world model
Noise-critic distillation‡	1 step per update
Target update rate τ	0.01 / 0.005
Critic (value) ensemble	10 Q^A + 10 Q^W, min / 3 nets, mean
Gradient clipping	10
Value initialisation	RoboMeter progress (offline)
Online progress distillation†	every 20 value updates[a]
Data	
Replay buffer (chunks)	100,000 / 60,000
Warm-up	1,000 chunks (≈8k env steps) with prior noise, no updates
Demonstration : online sampling	$1.0 \rightarrow 0.1$ over the first 40k env steps, then 0.1
Exploration	actor noise w.p. $\min(1, t/75\text{k})$, otherwise prior noise
Total env steps	300k
Networks	
Cameras; resolution	agentview + wrist; 224×224
Actor encoder	IMPALA, 128 per camera
DrQ random shift	4 px on critic inputs (s, s') / 4 px on actor inputs only
Critic (value) input	IMPALA / frozen DINOv2-S/14 + per-camera 4-query attention readout[b]
World model†	DINO-WM (6 layers, 6 heads, width 384)
MLPs; proprioception	3×256; 9-D

[a] Square_d0 only; the other tasks use TD only. [b] Square_d1 and threading: readout pretrained on segment-level RoboMeter progress.

B Additional Ablation Study

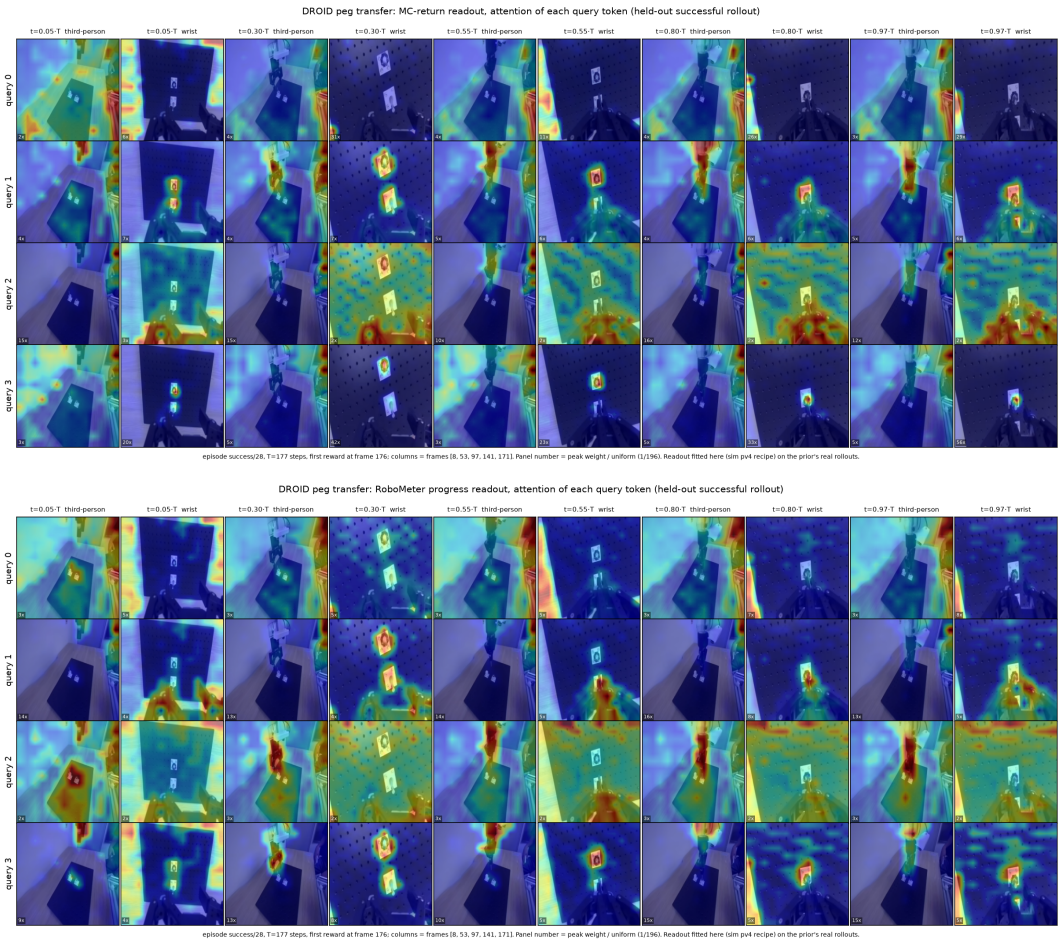


Figure 6: **Attention-pooling visualizations for Peg Insertion.** Top: attention from the value function pretrained using Monte Carlo returns. Bottom: attention from the value function distilled from RoboMeter task-progress estimates. The RoboMeter-distilled value function attends more strongly to task-relevant visual features.

What does progress distillation change in the value representation? We visualize the value readout's attention (Appendix A.3) after pretraining on RoboMeter progress or on Monte Carlo (MC) returns. Both use the same architecture and data; only the target differs.

On real-robot Peg Transfer, the progress-distilled readout tracks the arm, peg and target in both views. The MC readout concentrates on a few wrist-view spots, including a static wall edge. Measured against its own target, the progress readout also ranks held-out frames better (Spearman 0.76 vs. 0.43). We attribute this to the denser supervision. Progress is graded on failed episodes too, whereas MC returns are zero on every frame of a failure.