A Further Details on the Experimental Setup

A.1 Task Descriptions

We consider a total of 8 continuous control tasks from 2 benchmarks: ManiSkill (Mu et al., 2021), and Adroit (Rajeswaran et al., 2017). This section provides detailed task descriptions on overall information, task difficulty, object sets, state space, and action space. Some task details are listed in Table 2.

A.1.1 ManiSkill Tasks

For all tasks we evaluated on ManiSkill benchmark, we use consistent setup for state space, and action space. The state spaces adhere to a standardized template that includes proprioceptive robot state information, such as joint angles and velocities of the robot arm, and, if applicable, the mobile base. Additionally, task-specific goal information is included within the state. ManiSkill tasks we evaluated are very challenging because two of them require precise control and another two involve object variations. Below, we present the key details pertaining to the tasks used in this paper.

Stack Cube

- Overall Description: Pick up a red cube and place it onto a green one.
- Task Difficulty: This task requires precise control. The gripper needs to firmly grasp the red cube and accurately place it onto the green one.
- Object Variations: No object variations.
- Action Space: Delta position of the end-effector and joint positions of the gripper.
- State Observation Space: Proprioceptive robot state information, such as joint angles and velocities of the robot arm, and task-specific goal information.
- Visual Observation Space: one 64x64 RGBD image from a base camera and one 64x64 RGBD image from a hand camera.

Peg Insertion Side

- Overall Description: Insert a peg into the horizontal hole in a box.
- Task Difficulty: This task requires precise control. The gripper needs to firmly grasp the peg, perfectly aligns it horizontally to the hole, and inserts it.
- Object Variations: The box geometry is randomly generated
- Action Space: Delta pose of the end-effector and joint positions of the gripper.
- State Observation Space: Proprioceptive robot state information, such as joint angles and velocities of the robot arm, and task-specific goal information.
- Visual Observation Space: one 64x64 RGBD image from a base camera and one 64x64 RGBD image from a hand camera.

Turn Faucet

- Overall Description: Turn on a faucet by rotating its handle.
- Task Difficulty: This task needs to handle object variations. The dataset contains trajectories of 10 faucet types, while in online interactions, the agent needs to deal with 3 novel faucets not present in the dataset. See Fig 12.
- Object Variations: We have a source environment containing 10 faucets, and the dataset is collected in the source environment. The agent interacts with the target environment online, which contains 3 novel faucets.
- Action Space: Delta pose of the end-effector and joint positions of the gripper.
- State Observation Space: Proprioceptive robot state information, such as joint angles and velocities of the robot arm, the mobile base, and task-specific goal information.

- Visual Observation Space: one 64x64 RGBD image from a base camera and one 64x64 RGBD image from a hand camera.

Push Chair

- Overall Description: A dual-arm mobile robot needs to push a swivel chair to a target location on the ground (indicated by a red hemisphere) and prevent it from falling over. The friction and damping parameters for the chair joints are randomized.
- Task Difficulty: This task needs to handle object variations. The dataset contains trajectories of 5 chair types, while in online interactions, the agent needs to deal with 3 novel chairs not present in the dataset. See Fig 12.
- Object Variations: We have a source environment containing 5 chairs, and the dataset is collected in the source environment. The agent interacts with the target environment online, which contains 3 novel chairs.
- Action Space: Joint velocities of the robot arm joints and mobile robot base, and joint positions of the gripper.
- State Observation Space: Proprioceptive robot state information, such as joint angles and velocities of the robot arm, task-specific goal information.
- Visual Observation Space: three 50x125 RGBD images from three cameras 120° apart from each other mounted on the robot.

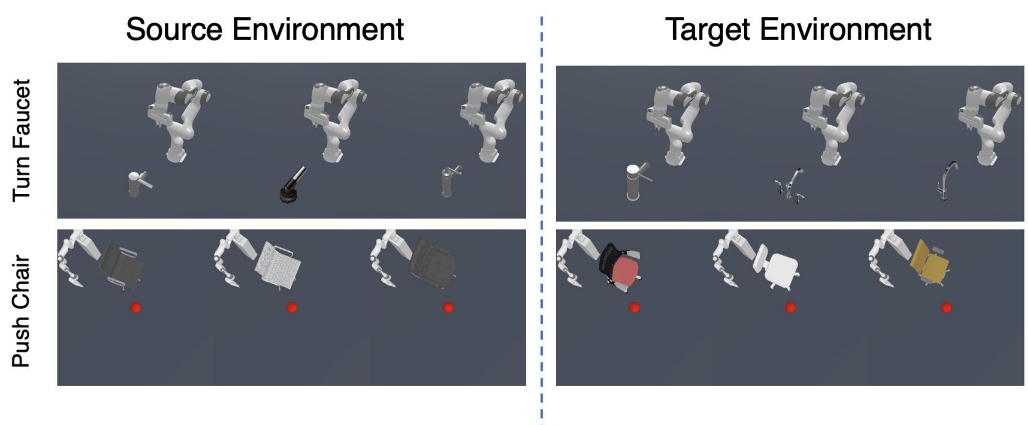

Figure 12: For the Turn Faucet and Push Chair tasks in the ManiSkill benchmark, *we have a source environment with various object variations from which the dataset is collected. The agent interacts with a target environment that features novel object variations.* Please refer to the information above for specific details.

A.1.2 Adroit Tasks

Adroit Door

- Overall Description: The environment is based on the Adroit manipulation platform, a 28 degree of freedom system which consists of a 24 degrees of freedom ShadowHand and a 4 degree of freedom arm. The task to be completed consists on undoing the latch and swing the door open.
- Task Difficulty: The latch has significant dry friction and a bias torque that forces the door to stay closed. No information about the latch is explicitly provided. The position of the door is randomized.
- Object Variations: No object variations.
- Action Space: Absolute angular positions of the Adroit hand joints.

- State Observation Space: The angular position of the finger joints, the pose of the palm of the hand, as well as state of the latch and door.
- Visual Observation Space: one 128x128 RGB image from a third-person view camera.

Adroit Pen

- Overall Description: The environment is based on the Adroit manipulation platform, a 28 degree of freedom system which consists of a 24 degrees of freedom ShadowHand and a 4 degree of freedom arm. The task to be completed consists on repositioning the blue pen to match the orientation of the green target.
- Task Difficulty: The target is also randomized to cover all configurations.
- Object Variations: No object variations.
- Action Space: Absolute angular positions of the Adroit hand joints.
- State Observation Space: The angular position of the finger joints, the pose of the palm of the hand, as well as the pose of the real pen and target goal.
- Visual Observation Space: one 128x128 RGB image from a third-person view camera.

Adroit Hammer

- Overall Description: The environment is based on the Adroit manipulation platform, a 28 degree of freedom system which consists of a 24 degrees of freedom ShadowHand and a 4 degree of freedom arm. The task to be completed consists on picking up a hammer with and drive a nail into a board.
- Task Difficulty: The nail position is randomized and has dry friction capable of absorbing up to 15N force.
- Object Variations: No object variations.
- Action Space: Absolute angular positions of the Adroit hand joints.
- State Observation Space: The angular position of the finger joints, the pose of the palm of the hand, the pose of the hammer and nail, and external forces on the nail.
- Visual Observation Space: one 128x128 RGB image from a third-person view camera.

Adroit Relocate

- Overall Description: The environment is based on the Adroit manipulation platform, a 30 degree of freedom system which consists of a 24 degrees of freedom ShadowHand and a 6 degree of freedom arm. The task to be completed consists on moving the blue ball to the green target.
- Task Difficulty: The positions of the ball and target are randomized over the entire workspace.
- Object Variations: No object variations.
- Action Space: Absolute angular positions of the Adroit hand joints.
- State Observation Space: The angular position of the finger joints, the pose of the palm of the hand, as well as kinematic information about the ball and target.
- Visual Observation Space: one 128x128 RGB image from a third-person view camera.

Table 2: We consider 8 continuous control tasks from 2 benchmarks. We list important task details below.

Task	State Observation Dim	Action Dim	Max Episode Step
ManiSkill: StackCube	55	4	200
ManiSkill: PegInsertionSide	50	7	200
ManiSkill: TurnFaucet	43	7	200
ManiSkill: PushChair	131	20	200
Adroit: Door	39	28	300
Adroit: Pen	46	24	200
Adroit: Hammer	46	26	400
Adroit: Relocate	39	30	400

A.2 Demonstrations

This subsection provides the details of demonstrations used in our experiments. See Table 3. ManiSkill demonstrations are provided in Gu et al. (2023), and Adroit demonstrations are provided in Rajeswaran et al. (2017).

Table 3: We list the number of demonstrations and corresponding generation methods below.

Task	Num of Demo Trajectories	Generation Method
ManiSkill: StackCube	1000	Task and Motion Planning
ManiSkill: PegInsertionSide	1000	Task and Motion Planning
ManiSkill: TurnFaucet	1000	Model Predictive Control
ManiSkill: PushChair	1000	Reinforcement Learning
Adroit: Door	25	Human Teleoperation
Adroit: Pen	25	Human Teleoperation
Adroit: Hammer	25	Human Teleoperation
Adroit: Relocate	25	Human Teleoperation

B Implementation Details

B.1 Base Policies

We experiment with 2 state-of-the-art imitation learning models: Behavior Transformer and Diffusion Policy.

B.1.1 Behavior Transformer

We follow the setup of Behavior Transformer in the original paper (Shafiullah et al., 2022). The architecture hyperparameters are included in Table 4, and the training hyperparameters are included in Table 5.

Table 4: We list the important architecture hyperparameters of Behavior Transformer used in our experiments.

Hyperparameter	Value
Context Window	10/20
Num Clusters	4/8
Num Layers	4
Num Heads	4
Embedding Dimensions	128
Trainable Parameters	approximately 1 Million

Table 5: We list the important training hyperparameters of Behavior Transformer in ManiSkill and Adroit tasks below.

Hyperparameter	Value (ManiSkill)	Value (Adroit)
Gradient Steps	200000	5000
Batch Size	2048	2048
Learning Rate	1e-4	1e-4
Evaluation Frequency	100 episodes every 5000 steps	100 episodes every 100 steps
Optimizer	AdamW Optimizer	AdamW Optimizer

B.1.2 Diffusion Policy

We follow the setup of U-Net version of Diffusion Policy in the original paper (Chi et al., 2023). The architecture hyperparameters are includes in Table 6, and the training hyperparameters are included in Table 7.

Table 6: We list the important architecture hyperparameters of Diffusion Policy used in our experiments.

Hyperparameter	Value
Action Horizon	4
Observation Horizon	2
Prediction Horizon	16
Embedding Dimensions	64
Downsampling Dimensions	256, 512, 1024
Trainable Parameters	approximately 4 Million

Table 7: We list the important training hyperparameters of Diffusion Policy in ManiSkill and Adroit tasks below.

Hyperparameter	Value (ManiSkill)	Value (Adroit)
Gradient Steps	200000	200000
Batch Size	1024	1024
Learning Rate	1e-4	1e-4
Evaluation Frequency	100 episodes every 5000 steps	100 episodes every 5000 steps
Optimizer	AdamW Optimizer	AdamW Optimizer

B.1.3 Checkpoint Selection

We evaluate the base policy for 50 episodes every specific number of gradient steps during training. We select the checkpoint with the highest evaluation success rate.

B.2 Policy Decorator (Our Approach)

Policy Decorator framework introduces two key hyperparameters: H **in Progressive Exploration Schedule** and **Bound** α **of Residual Actions**. We list the values of these two key hyperparameters across all tasks in the table below. Both of them are not too difficult to tune. We typically set α close to the action scale observed in the demonstration dataset and make minor adjustments. H has a wide workable range, and using a large H is generally a safe choice if sample efficiency is not the primary concern. See Section 5.4.2 for more disccusion on the influence of these two hyperparameters.

Table 8: The values of H in Progressive Exploration Schedule and Bound α of Residual Actions across all tasks.

Task	H	α
ManiSkill: StackCube (BeT, state)	1M	0.03
ManiSkill: PegInsertionSide (BeT, state)	1M	1.0
ManiSkill: TurnFaucet (BeT, state)	500K	0.2
ManiSkill: PushChair (BeT, state)	4M	0.2
Adroit: Door (BeT, state)	100K	0.3
Adroit: Pen (BeT, state)	100K	0.3
Adroit: Hammer (BeT, state)	100K	0.3
Adroit: Relocate (BeT, state)	100K	0.2
ManiSkill: PegInsertionSide (Diffusion Policy, state)	30K	0.03
ManiSkill: TurnFaucet (Diffusion Policy, state)	100K	0.1
ManiSkill: PushChair (Diffusion Policy, state)	100K	0.2
Adroit: Pen (Diffusion Policy, state)	100K	0.2
Adroit: Hammer (Diffusion Policy, state)	100K	0.1
Adroit: Relocate (Diffusion Policy, state)	300K	0.1
ManiSkill: TurnFaucet (Diffusion Policy, visual)	30K	0.05
ManiSkill: PushChair (Diffusion Policy, visual)	100K	0.2
Adroit: Door (Diffusion Policy, visual)	1M	0.1
Adroit Pen (Diffusion Policy, visual)	100K	0.8

B.3 Important Shared Hyperprameters among Policy Decorator and other Baselines

As all baselines use SAC as the backbone RL algorithm, we include some important shared hyper-parameters used among the Policy Decorator and baselines in our experiments. See the Table 9 for more details.

Table 9: We list the important shared hyperparameters among Policy Decorator and other baselines in ManiSkill and Adroit tasks below.

Hyperparameter	Value (ManiSkill)	Value (Adroit)
Gamma	0.90	0.97
Batch Size	1024	1024
Learning Rate	1e-4	1e-4
Policy Update Frequency	1	1
Training Frequency	64	64
Update-to-data Ratio	0.25	0.25
Target Network Update Frequency	1	1
Tau	0.01	0.01
Learning Starts	8000	8000

B.4 Enable RL Fine-tuning on Base Policies

B.4.1 SAC for Behavior Transformer

Special Modifications on BeT Special adaptations relate to SAC's Gaussian Tanh Policy, which requires the actor backbone to output in the ATANH space of action rather than the regular space. This requirement complicates the initialization of the Behavior Transformer (BeT) as the actor backbone. Therefore, we allow the clustering process in BeT to operate in the regular action space, but the regression head outputs in the ATANH action space. The final action is then computed as:

$$\mathbf{a}_{\text{final}} = \text{arctanh}(\mathbf{a}_{\text{bin}}) + \mathbf{a}_{\text{regression output}}$$

Since the atanh function is defined between -1 and 1, some action dimensions (e.g., gripper actions) need to be scaled to avoid numerical issues. In ManiSkill, we multiply the gripper dimension (last action dimension) by 0.3; in Adroit, we multiply all actions by 0.5. The actions are rescaled back after going through tanh. Our BeT, specially modified for fine-tuning, achieves similar performance in evaluations in order to enable fair comparison. See Table 10 for evaluation success rate of BeT and BeT modified version in ManiSkill and Adroit tasks.

Following the general paradigm of fine-tuning GPT-based models in natural language processing, we add LoRA to all attention layers and final regression heads.

Table 10: We list the evaluation success rate of BeT and BeT modified version in ManiSkill and Adroit tasks. BeT modifiled version is used in fine-tuning baselines, and original BeT is used in Policy Decorator and non-fine-tuning baselines.

Task	BeT	BeT modified version
ManiSkill: StackCube (state)	71%	67%
ManiSkill: PegInsertionSide (state)	15%	13%
ManiSkill: TurnFaucet (state)	41%	35%
ManiSkill: PushChair (state)	18%	23%
Adroit: Door (state)	78%	77%
Adroit: Pen (state)	65%	63%
Adroit: Hammer (state)	23%	21%
Adroit: Relocate (state)	20%	13%

Special Modifications on SAC We use SAC as our primary fine-tuning algorithm for Behavior Transformer, with actor initialized using a pre-trained Behavior Transformer and a MLP as Q function. See Appendix F.5.1 for discussion on the architecture choice of Q function.

B.4.2 DIPO FOR DIFFUSION POLICY

Special Modifications on DIPO DIPO uses action gradients to optimize the actions, and convert online training to supervised learning, also refer to H.2. Since the Diffusion Policy employs a prediction horizon that exceeds the action horizon (receding horizon), during the DIPO training phase, we focus on optimizing only the first action horizon within the total prediction horizon using action gradients. This approach prevents dynamics inconsistencies that would arise from optimizing the remaining actions.

Following the general paradigm of fine-tining diffusion-based models in visual, we add LoRA to all layers of diffusion policy.

B.5 BASELINES

In our experiments, we compare Policy Decorator with several strong baseline methods. The following section provides implementation details for these baseline approaches.

Basic RL See Appendix B.4.

Regularized Optimal Transport (ROT) (Behavior Transformer Only). ROT (Haldar et al., 2023a) is an online fine-tuning algorithm that fine-tunes a pre-trained base policy using behavior cloning (BC) regularization with adaptive Q-filtering and optimal transport (OT) rewards. We use pre-trained Behavior Transformer as base policy. For Behavior Cloning regularization, we allow BeT to output the entire window of actions and apply the regularization accordingly. In experiments involving state observations, the optimal transport (OT) rewards are computed using a 'trunk' network within the value function, which consists of a single-layer neural network. In contrast, for experiments with visual observations, the OT rewards are computed directly using the visual encoder network. The other experimental setup follows SAC.

Reinforcement Learning with Prior Data (RLPD) (Behavior Transformer Only). RLPD (Ball et al., 2023) is a state-of-the-art online learn-from-demo method that enhances the vanilla SACfd with critic layer normalization, symmetric sampling, and sample-efficient RL (Q ensemble + high UTD). We add layer normalization to critic network. We maintain one offline buffer, which includes demonstration data, and one online buffer, which contains online data. For online updates, we sample 50% batch from offline buffer and 50% batch from online buffer. We omit the sample-efficient RL (Q ensemble + high UTD) due to the significant training costs associated with these components and to ensure a fair comparison with other methods. The omitted component pursues extreme sample efficiency at the cost of significantly increased wall-clock training time, which is impractical, especially when fine-tuning a large model. The other experiment setup follows SAC.

Calibrated Q-Learning (Cal-QL) (Behavior Transformer Only). Cal-QL (Nakamoto et al., 2024) is an offline RL online fine-tuning method that "calibrates" the Q function of vanilla CQL. We pre-train a Q function using Cal-QL in the offline stage and then use SAC for fine-tuning in the online stage with this pre-trained value function. We opted for this offline-to-online strategy because, in the online stage of the original Cal-QL paper, calculating the critic loss requires querying the actor 20 times. This process is time-intensive, especially considering that the actor is initialized as a large base model. The performance of curve C in Fig. 22 demonstrates the effectiveness of this strategy. See F.3 for more discussion. In offline stage, we use pre-trained BeT with gradients open as actor and an MLP as critic. In online stage, we use pre-trained BeT as actor and offline-trained MLP as critic. The other experiment setup follows SAC.

Jump-Start Reinforcement Learning (JSRL) (Both Behavior Transformer and Diffusion Policy). JSRL (Uchendu et al., 2023) is a curriculum learning algorithm that uses an expert teacher policy to guide the student policy. In our setting, we use a pre-trained large policy (BeT or diffusion policy) as the guiding policy and an MLP as the online actor. The initial jump start steps are the average length of success trajectories in 100 evaluations of the pre-trained base policy. Following the setup in the original paper, we maintain a moving window of evaluation success rate and best moving average success rate. If current moving evaluation success rate is within the range of [best moving average - tolerance, best moving average + tolerance], then we go 10 steps backwards.

Residual Reinforcement Learning (Residual RL) (Both Behavior Transformer and Diffusion Policy). Residual RL (Johannink et al., 2019) learns a residual policy in an entirely uncontrolled manner. In our experiments, We use a pre-trained large policy as the base policy and a small MLP as the online residual actor. We follow the setting in the original paper that in online interactions, final action = base action + online residual action.

Fast Imitation of Skills from Humans (FISH) (Both Behavior Transformer and Diffusion Policy). FISH (Haldar et al., 2023b) builds upon Residual RL by incorporating a non-parametric nearest neighbor search VINN policy (Pari et al., 2021) and learning an online offset actor with optimal transport rewards. In our experiments, we use a GPT backbone as the representation network for BeT experiments, a FiLM encoder (Perez et al., 2018) for diffusion state observation mode experiments, and a visual encoder for visual observation mode experiments. See Appendix G.2.1 for the performance of VINN policy.

C Additional Results of Policy Decorator

C.1 The Performance of RL from Scratch

The RL training from scratch baseline has been incorporated into Fig. 13. We only plot results on Adroit, as RL training from scratch achieves 0% success rate on ManiSkill tasks.

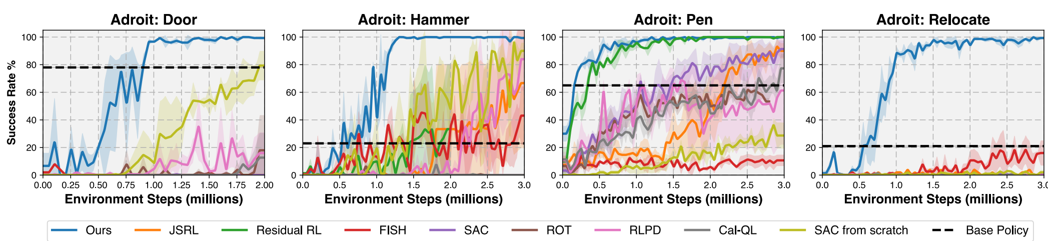

Figure 13: Add SAC (training from scratch) to Fig. 6. Results are only shown for Adroit tasks, as it achieves 0% success rate on all ManiSkill tasks with sparse reward.

C.2 Comparison with DPPO

C.2.1 Setup

DPPO (Ren et al., 2024), a very recent work, successfully fine-tunes diffusion policies using PPO, achieving state-of-the-art performance. Key tricks include fine-tuning only the last few denoising steps and fine-tuning DDIM sampling. Given that **this project was released around three weeks before the ICLR deadline**, we lacked sufficient time to fully adapt it to our tasks. Nevertheless, we conducted preliminary experiments comparing our approach with DPPO **on their tasks**. Even if DPPO is carefully tuned on their tasks, we are still able to beat it.

Specifically, we applied Policy Decorator (our approach) to the **two most challenging robotic manipulation tasks in their paper**: Square and Transport. We used the Diffusion Policy checkpoints provided by the DPPO paper as our base policies.

C.2.2 RESULTS

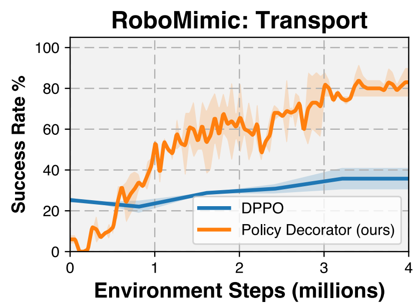

Figure 14: Policy Decorator (ours) vs. DPPO on the Transport task.

As shown in Fig. 14, our approach significantly outperforms DPPO on the Transport task. According to Figure 5 in the DPPO paper, DPPO requires approximately 16 million steps to converge to 80%+ success rate on the Transport task. In contrast, our Policy Decorator achieves this performance in only 4 million steps, demonstrating a **nearly 4x improvement in sample efficiency**.

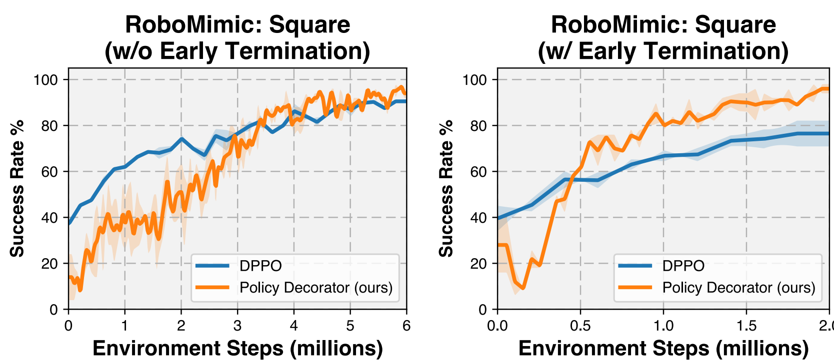

Figure 15: Policy Decorator (ours) vs. DPPO on the Square task.

On the Square task, our approach performs comparably to DPPO (left subfigure in Fig. 15). Upon further investigation, we discovered that DPPO uses a fixed episode length without early termination upon success signals. Empirically, this setup may negatively impact the sample efficiency of RL algorithms, as transitions after task completion contribute minimally to learning. Consequently, we conducted an additional experiment implementing early task termination upon success signals. **The results (right subfigure in Fig. 15) demonstrate that our approach outperforms DPPO in this more reasonable setup.**

C.2.3 SUMMARY

These experiments demonstrate that our method outperforms DPPO on challenging robotic manipulation tasks. It is crucial to note that our approach is model-agnostic, whereas DPPO is restricted to a specific case of Diffusion Policy (where all predicted actions are executed in the environment, which is not the typical implementation of Diffusion Policy).

D ADDITIONAL ABLATION STUDIES

This section includes additional ablation studies results about base policies, low-performing checkpoints, and PPO. In detail, Section D.1 discusses Policy Decorator also works with other types of base policies (e.g., MLP, RNN, and CNN); Section D.2 demonstrates that Policy Decorator stays effective in improving low-performing BeT checkpoints; Section D.3 indicates that Policy Decorator is compatible with PPO as backbone RL algorithm.

D.1 Additional Base Policies

To demonstrate that Policy Decorator is truly versatile to all types of base policy, we further experiment with model architecture of low representation power like MLP, BC-RNN, and CNN as well as low performance checkpoints of Behavior Transformer.

Fig. 16 demonstrates that the Policy Decorator significantly enhances the performance of MLP, BC-RNN, and CNN policies by interacting with environments.

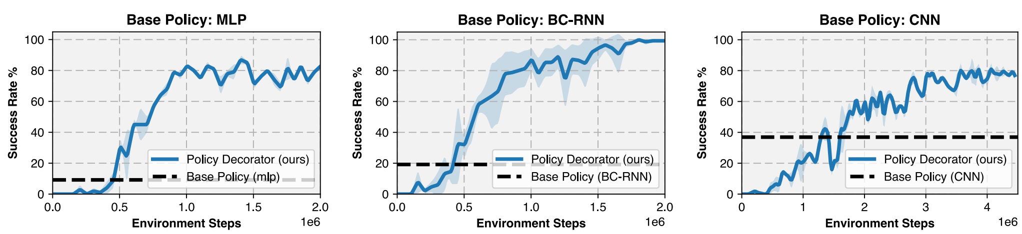

Figure 16: Policy Decorator with more base policies (MLP, BC-RNN, CNN) on TurnFaucet task through online interactions.

D.2 Using Other Checkpoints of Base Policies

As we claim that Policy Decorator is model-agnostic and is versatile to all types of base policies, it is necessary to demonstrate that it not only improves well-trained base policy but also improves low-performing checkpoints of base policy. Fig. 17 shows that the Policy Decorator achieves a substantial improvement in the low-performance BeT checkpoint.

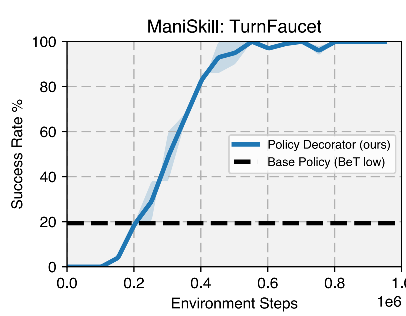

Figure 17: Policy Decorator with a low-performance BeT checkpoint.

D.3 Change Backbone RL Algorithm to PPO

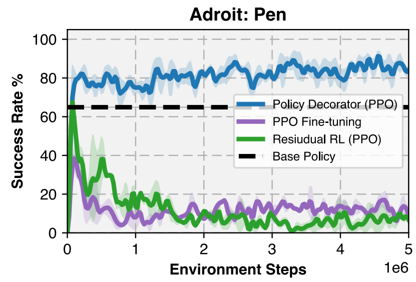

Figure 18: Use PPO as the backbone RL algorithm in our method, RL fine-tuning, and Residual RL.

While we use SAC as the backbone RL algorithm in our experiments due to its high sample efficiency, it is essential to demonstrate that the Policy Decorator can be integrated with other categories of RL algorithms, such as policy optimization, to provide greater flexibility. We changed backbone RL algorithm of our method, RL fine-tuning baseline, and residual RL baseline from SAC to PPO (Schulman et al., 2017). As shown in Fig. 18, Policy Decorator with PPO successfully improves the base policy and considerably outperforms all baselines.

E Important Design Choices

This section presents ablation results on a few key design choices, including the inputs for the residual policy and the inputs for the critic.

E.1 Input of Residual Policy

The residual policy can receive input in the form of either observation alone or both observation and action from the base policy. Our experiments indicate that using only observation typically produces better results, as illustrated in Fig. 19.

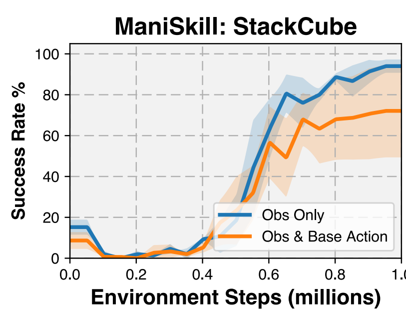

Figure 19: Different variants of input of residual policy.

E.2 Input of Critic

In SAC, the critic $Q(s, a)$ takes an action as input, and there are several design choices regarding this action: we can use 1) the sum of the base action and residual action; 2) the concatenation of both; or 3) the residual action alone. Based on our experiments shown in Fig. 20, using the sum of both actions yields the best performance.

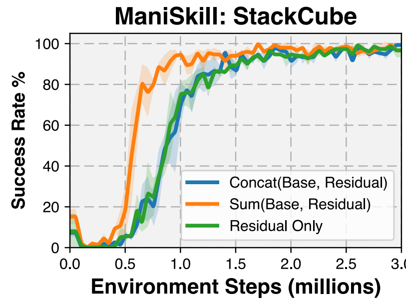

Figure 20: Different variants of input of critic.

F FAILURE OF FINE-TUNING BASELINES

In this section, we analyze the poor performance of fine-tuning baselines in our experiments. **We provide an overall explanation for these failures in Sec. F.1. Then, Sec. F.2, F.3, and F.4 offer illustrative experiments supporting the arguments presented in Sec. F.1.** Finally, Sec. F.5 presents some additional ablation studies on design choices in fine-tuning baselines, demonstrating our careful tuning of baseline implementations to achieve better performance.

F.1 OVERALL EXPLANATION

Even if we have selected the strongest learning-from-demo methods, most of them are still not specifically designed for fine-tuning, and they do not intentionally prevent the unlearning of the base model, i.e., the performance can drop significantly at the very beginning of training. This phenomenon has also been discussed in Nakamoto et al. (2024). According to our observations, we believe that performance degradation is probably due to the following two reasons:

1. **Random Critic Initialization:** We believe the randomly initialized critic network cannot provide meaningful gradients to guide the policy. Such a noisy gradient can easily cause the policy to deviate significantly from the initial weights. Once the unlearning happens, it becomes very hard to relearn the policy since it cannot get the sparse reward signal anymore. Sec. F.2 presents an illustrative experiment to show this policy degradation with randomly initialized critic. On the other hand, Cal-QL (Nakamoto et al., 2024) can theoretically learn a critic from offline data. However, our empirical results indicate that when trained purely on demonstration data without negative trajectories, the learned critic does not significantly improve online fine-tuning. This performance degradation during Cal-QL online training aligns with observations reported by (Yang et al., 2023a). Experimental evidence supporting this analysis is presented in Sec. F.3.
2. **Long Task Horizon:** Long task horizon also significantly increases the difficulty of fine-tuning, particularly in sparse reward settings. As the task horizon increases, the agent's likelihood of discovering sparse rewards through random exploration diminishes. Additionally, the sparse reward signal requires more time to propagate through longer trajectories. The experiments presented in Sec. F.4 empirically validate that the long task horizon is a key factor contributing to the failure of fine-tuning baselines.

F.2 POLICY DEGRADATION WITH RANDOM INITIALIZED CRITIC

This section presents illustrative experiments demonstrating how updating the base policy with a randomly initialized critic function $Q(s, a)$ results in significant deviations from its original trajectory.

In the StackCube task, a robot arm must pick up a red cube and stack it on a green cube. Initially, a pre-trained base policy (Behavior Transformer) successfully grasps the red cube and accurately places it on the green cube, as shown in this video.

After fine-tuning the base policy with a randomly initialized critic for 100 gradient steps, the policy begins to deviate slightly from the original trajectory, as shown in this video. While still able to grasp the red cube, it fails to precisely place it on the green cube.

Following an additional 100 updates (200 total), the base policy deviates further from the original trajectory, struggling to effectively grasp the red cube, as shown in this video.

In summary, these experiments suggest that fine-tuning the base policy with a randomly initialized critic can lead to unlearning. Once unlearning occurs, it becomes very hard to relearn the policy since it cannot get the sparse reward signal anymore.

F.3 Pre-training Critic on Demo-only Dataset Does Not Help

Cal-QL (Nakamoto et al., 2024), a state-of-the-art offline RL method, aims to pre-train a critic for efficient online fine-tuning. Our experiments show that pre-training a critic using Cal-QL on demonstration-only datasets (without negative experiences) provides limited benefits for online fine-tuning, as illustrated in Fig. 21. **This section presents experiments explaining why it does not help and validates the correctness of our Cal-QL baseline results.**

The original Cal-QL paper reported much better results on Adroit tasks compared to our Cal-QL baseline. We believe this discrepancy is mainly due to differences in experimental setups:

1. **Offline Dataset:** The original Cal-QL paper uses an offline dataset consisting of 25 human teleoperation demonstrations and additional trajectories from a BC policy. Our Cal-QL baseline uses only 25 human demonstrations, ensuring fair comparison with other learning-from-demo baselines that only utilize demonstrations. We also made this assumption in Sec. 3.
2. **Actor Architecture:** The original Cal-QL paper employs a small MLP as the actor, while we use a pre-trained Behavior Transformer (BeT) to align with our goal of improving the pre-trained base policy.
3. **Online Algorithm:** The original Cal-QL paper uses Cal-QL algorithm in both offline and online stage. However, computing critic loss in Cal-QL algorithm requires querying the actor 20 times in each update, which is extremely time-consuming given that the actor is a large model in our settings. Therefore, we use SAC in the online phase instead of Cal-QL.

To verify whether these setup differences cause the divergent results, we designed the following experimental setups for Cal-QL, **interpolating between the original setup and ours:**

- A: Small MLP actor + Mixed dataset + online Cal-QL (Cal-QL's original setting)
- B: Small MLP actor + Demo-only dataset + online Cal-QL
- C: Small MLP actor + Demo-only dataset + online SAC
- D: Large GPT actor + Demo-only dataset + online SAC
- E: BeT actor + Demo-only dataset + online SAC (the setup used in our experiments)

The experimental results of these setups are shown in Fig. 22. **In Cal-QL's paper, they only report the results up to 300k steps, and our curve A perfectly matches the official results, which suggests that our implementation is correct.** Interestingly, Cal-QL exhibits instability when run for longer periods (e.g., 3M steps), even in its original setup. Comparing curve A and curve B illustrates Cal-QL's strong dependence on a large, diverse dataset comprising both demonstrations and negative trajectories. Cal-QL's sample efficiency deteriorates a lot when the offline dataset is limited to a few demonstrations without negative trajectories. The comparison between curve B and curve C demonstrates that while using SAC as an online algorithm results in slightly reduced sample efficiency, it still achieves 90%+ success rates. This trade-off suggests that *sacrificing a little bit of sample efficiency is acceptable in exchange for significant wall-clock time savings*. The comparison between curve C and curve D illustrates that a large GPT actor can also negatively impact Cal-QL's performance. Curve D and curve E demonstrate that using a pre-trained BeT outperforms a randomly initialized GPT, which is expected.

In conclusion, the divergent results between Cal-QL's original paper and our baseline can be attributed to different experimental setups. Our results are validated and reliable.

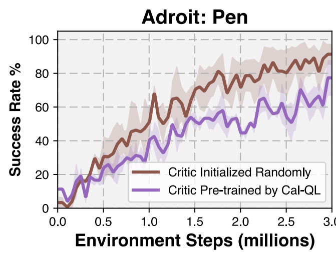

Figure 21: Pre-training a critic by Cal-QL on demo-only datasets does not help online fine-tuning.

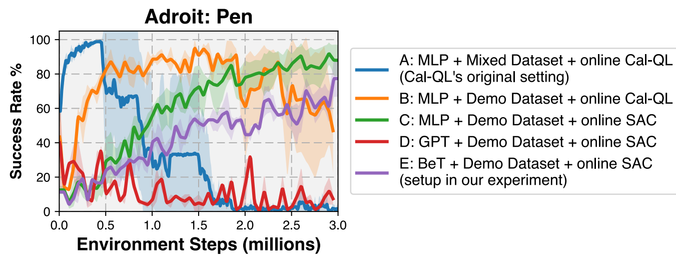

Figure 22: To verify whether the setup differences cause the divergent results, we designed different experimental setups for Cal-QL, interpolating between the original setup and ours.

F.4 Long Task Horizon Makes Fine-tuning Hard

This section presents experiments exploring how task horizon affects the fine-tuning of the base policy.

In the TurnFaucet task, no fine-tuning baselines achieve non-zero success rates. To shorten the effective task horizon, we roll out the pre-trained base policy (Behavior Transformer) for a specific number of steps (40, 100, or 120) in each episode. This approach likely brings the agent closer to success, thus shortening the effective task horizon. We then perform regular RL fine-tuning for the remaining steps of an episode.

Fig. 23 demonstrates that shortening the task horizon by 100 steps results in a significant improvement, while reducing it by 120 steps achieves a 100% success rate. This experiment clearly shows that the **long task horizon is a major factor in fine-tuning failure, and reducing the task horizon substantially eases RL fine-tuning difficulties.**

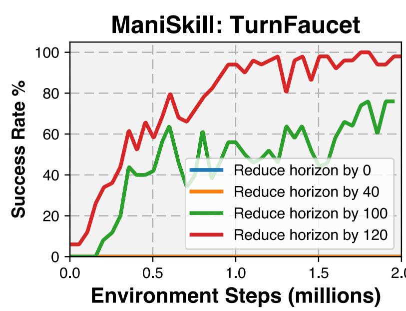

Figure 23: Fine-tuning Behavior Transformer using SAC with different effective task horizons.

F.5 Ablation Study on Design Choices in Fine-tuning Baselines

This section contains ablation studies on some design choices in fine-tuning-based baselines. In detail, Section F.5.1 discusses different choices of Q function architecture, while Section F.5.2 illustrates the effects of using warmstart in Q function training.

F.5.1 Architecture of Q Function

The architecture of the Q function can be important in designing fine-tuning baselines. We essentially have three options:

1. Q-function using an MLP
2. Q-function using a shared GPT backbone with the actor
3. Q-function using a separate GPT backbone

As shown in Fig. 24, we experimented with all the aforementioned Q-function architectures in SAC and PPO fine-tuning experiments. The results indicate that SAC fine-tuning with an MLP Q-function slightly improves the base policy, whereas SAC fine-tuning with the other two Q-function architectures does not yield such improvements. In contrast, PPO fine-tuning across all Q-function architectures demonstrates poor performance. Based on these observations, we chose to use the MLP Q-function in our fine-tuning baselines.

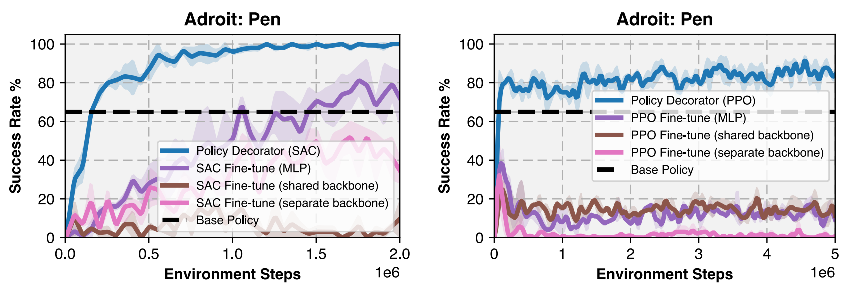

Figure 24: SAC/PPO fine-tuning with different critic architectures.

F.5.2 Effect of Warm-start in Q Function Training

Warm-starting Q function training is a widely used technique to ensure that the actor is updated with a reliable Q function. We also tried this technique in designing fine-tuning baselines. We experimented with a warm-start critic for a number of steps without training the actor. However, as shown in Fig. 25, this approach causes alpha, the learnable entropy coefficient in SAC, to increase massively, leading to an explosion in Q loss. We also compared vanilla fine-tuning with fine-tuning using a warm-start and fixed alpha. As indicated in Fig. 26, empirical results demonstrate that vanilla fine-tuning outperforms fine-tuning with a warm-start and fixed alpha. Upon closer examination, we found that fine-tuning with a warm-start and fixed alpha results in very unstable critic training. Therefore, we do not warm-start Q function training in our fine-tuning baselines.

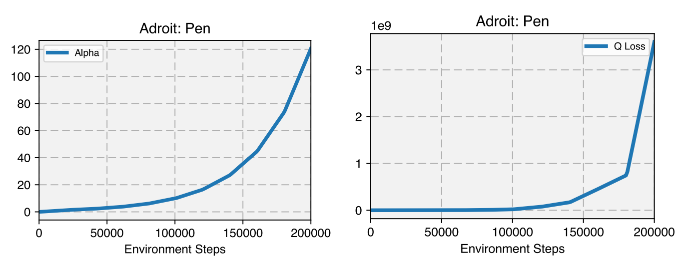

Figure 25: Critic warm start results in alpha and Q loss explosion when auto entropy tuning is enabled.

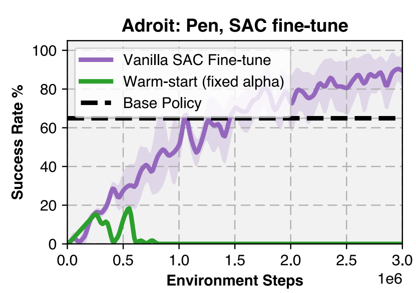

Figure 26: Warm-start the critic during fine-tuning.

G Failure of Non-fine-tuning Baselines

In this section, we analyze the poor performance of non-fine-tuning baselines in our experiments. We discusses the failure of vanilla Residual RL in Section G.1. We provides the explanations of failure of FISH in Section G.2.

G.1 Failure of Vanilla Residual RL

The residual RL baseline uses identical settings to our method, excluding the controlled exploration module. The primary failure mode of residual RL stems from 2 points:

1. Random residual actions in early training stages, causing the agent to deviate significantly from the base policy. This deviation leads to not getting any success signals for guiding learning. (see this video for an example).
2. Residual policy does not know it aims to minor fix the base policy, so during training, the average size of residual actions go beyond the average size of base policy actions, destroying the performance of base policy.

This is also supported by our ablation study (Fig. 10 and 11). As we gradually remove controlled exploration strategies (reducing H to 0 or increasing alpha to 1), our method approaches vanilla residual RL, resulting in deteriorating performance.

G.2 Failure of FISH

The primary failure mode of FISH stems from the extremely poor performance of non-parametric VINN policy in our experiments. See Section G.2.1 for the performance of VINN policy.

G.2.1 VINN Performance

The performance of VINN base policy are shown below.

Table 11: The performance of VINN base policy using GPT backbone from BeT under state observation.

Task	Success Rate
ManiSkill: StackCube	0%
ManiSkill: PegInsertionSide	0%
ManiSkill: TurnFaucet	1%
ManiSkill: PushChair	0%
Adroit: Door	12%
Adroit: Pen	16%
Adroit: Hammer	0%
Adroit: Relocate	2%

Table 12: The performance of VINN base policy using FiLM encoder from Diffusion Policy under state observation.

Task	Success Rate
ManiSkill: PegInsertionSide	0%
ManiSkill: TurnFaucet	0%
ManiSkill: PushChair	0%
Adroit: Pen	16%
Adroit: Hammer	0%
Adroit: Relocate	0%

Table 13: The performance of VINN base policy using visual encoder from Diffusion Policy under visual observation.

Task	Success Rate
ManiSkill: TurnFaucet	0%
ManiSkill: PushChair	0%
Adroit: Door	0%
Adroit: Pen	8%

H Fine-tuning Diffusion Policy using RL

H.1 Why Fine-tuning Diffusion Policy using RL is Non-trivial

Diffusion Models (Ho et al., 2020) and their applications in robotic control (Chi et al., 2023; Janner et al., 2022; Ajay et al., 2022) have traditionally been trained using supervised learning, where ground truth labels (e.g., images, actions) are required to supervise the denoising process.

Recently, novel approaches (Fan & Lee, 2023; Black et al., 2023; Uehara et al., 2024) have emerged, proposing the use of reinforcement learning (RL) to train diffusion models. The high-level idea involves modeling the denoising process as a Markov Decision Process (MDP) and assigning rewards based on the quality of the final denoised samples. This allows RL gradients to be backpropagated through the **inference process**, updating the model weights accordingly. This training paradigm represents a significant departure from conventional diffusion model training methods and **may face challenges when the number of denoising steps is large**. To date, these methods have primarily been applied in the domains of **image generation, molecule design, and DNA synthesis**.

However, **this training paradigm does not directly transfer to robotic control problems, particularly in sparse reward tasks**. As discussed in Ren et al. (2024), fine-tuning diffusion models in robotic control can be viewed as a "two-layer" MDP, where a complete denoising process with hundreds of steps represents a single decision step in the robotic control MDP. For example, if a robotic task requires 200 decision steps (actions) to complete, and a diffusion model uses 100 denoising steps to generate a decision (action), the reward in a sparse-reward robotic control task would be received only *every 20,000 denoising steps*. This presents a significantly greater challenge than training a diffusion model to generate images using RL, where rewards are typically received *every 100 denoising steps* under the same assumptions.

H.2 How "Basic RL for Diffusion Policy" Baseline is Selected

Despite the challenges in training diffusion policies for robotic control using RL, recent attempts have emerged. These can be broadly grouped into three categories. We will briefly explain each method and discuss the selection of the "Basic RL" baseline for fine-tuning diffusion policy.

Converting RL into Supervised Learning Methods in this category adhere to the conventional training recipe of the diffusion models, and try to define a "ground truth action label" for supervision. DIPO (Yang et al., 2023b) introduces "action gradient," using gradient descent on $Q(s, a)$ to estimate the optimal action for state s. **DIPO is selected as the basic RL algorithm in our experiments.** IDQL (Hansen-Estruch et al., 2023) constructs an implicit policy by reweighting samples from a diffusion-based policy, and using the implicit policy to supervise the training of the diffusion-based policy. We did not select it as the fine-tuning baseline for two reasons: 1) the training can be extremely slow especially with large base policies, because IDQL involves sampling the diffusion model multiple times (32 to 128 in their code) to compute the implicit policy; 2) as reported in its paper, IDQL performs worse than Cal-QL and RLPD, which are included in our baselines.

Matching the Score to the Q Function QSM (Psenka et al., 2023) aims to match the score Ψ of the diffusion-based policy to the gradient of the Q function $\nabla_a Q^\Psi(s, a)$ using supervised learning. According to Ren et al. (2024), QSM performs poorly in robotic manipulation tasks, thus it is not considered a competitive baseline.

Backpropagating RL Gradients Through the Inference Process Methods in this category adapt the training recipe discussed in H.1 to robotic control tasks, employing additional techniques to make it work. The actor's training objective is to maximize $Q(s, a)$. Diffusion QL (Wang et al., 2022) represents a basic version of these methods, primarily used in offline RL settings. However, its online performance is poor, as reported by Ren et al. (2024). Consistency AC (Ding & Jin, 2023) distills diffusion models into consistency models, significantly shortening the gradient propagation path. Nevertheless, its offline-to-online performance, as reported in its own paper, is even worse than Diffusion QL, thus we do not consider it a competitive baseline.

DPPO (Ren et al., 2024), a very recent work, successfully fine-tunes diffusion policies using PPO, achieving state-of-the-art performance. Key tricks include fine-tuning only the last few denoising steps and fine-tuning DDIM sampling. Given that **this project was released around three weeks before the ICLR deadline**, we lacked sufficient time to fully adapt it to our tasks. Nevertheless, we conducted preliminary experiments comparing our approach with DPPO *on their tasks*. Results indicate that our method significantly outperforms DPPO on their tasks. See Appendix C.2 for more details.