Validating Diffusion-Based Visual Imitation Learning for Robotic Manipulation

Guowei Huai
HKUST(GZ)
ghuai073@connect.hkust-gz.edu.cn

Jiahong Chen
HKUST(GZ)
jchen758chen@connect.hkust-gz.edu.cn

Yiming Zhu
HKUST(GZ)
yzhu714@connect.hkust-gz.edu.cn

1 Introduction

In recent years, diffusion models have emerged as a powerful class of generative models, demonstrating remarkable success in high-dimensional data generation tasks such as image and video synthesis. Their ability to capture complex, multimodal distributions has led to widespread adoption in the vision community, setting new benchmarks in generative quality and diversity. This surge of progress has sparked growing interest in exploring whether the same capabilities can be translated into other domains—particularly in robotics, where learning expressive, high-capacity policies remains a central challenge.

Motivated by this trend, we are interested in understanding how the generative strengths of diffusion models can be leveraged in the context of robot learning. Specifically, we are drawn to the problem of visuomotor policy learning, where policies must map from high-dimensional, perceptual inputs—such as images or point clouds—to continuous control actions. This task is inherently complex and multimodal, making it a promising target for generative modeling techniques.

In this project, we aim to investigate the potential of diffusion-based policies in robotic imitation learning. To do so, we conduct a systematic study and reproduction of two recently proposed approaches: 3D Diffusion Policy (DP3) [1] and Robotics Diffusion Transformer (RDT-1B) [2]. Through this exploration, we seek to better understand the practical benefits, limitations, and design choices involved in applying diffusion models to real-world robotic control problems.

2 Related Work

2.1 Teleoperation System

Supervised training with expert demonstration data, or imitation learning (IL), enables a machine to learn from the expert's decisions, replicating the expert's actions as closely as possible. As one of the most sample-efficient learning methods, IL has made great strides in enabling robots to mimic human-level dexterity in manipulation tasks. The collection of sufficient and practical expert demonstration data is a key factor in determining the success of an IL-based model.

As a result, recent research has increasingly focused on the use of various tools to capture human limb movement data to control robots for manipulation tasks, often referred to as teleoperation systems. Handheld controllers such as joysticks [3] are easy to set up, but their limited degrees of freedom make them unsuitable for precise and complex movements. Virtual reality (VR) headsets [4, 5] are highly interactive, and vision devices [6, 7, 8] such as cameras with varying settings can acquire more precise spatial information about humans and objects. However, they are susceptible to issues such

AIAA 5032 "Foundations of Artificial Intelligence" Course Project Report at HKUST(GZ).

as occlusion and lighting effects, especially during hand-object interactions, which are of particular interest to researchers. Also, the above methods are typically not intuitive enough for teleoperation and require extra training for the operator. To address these challenges, recent exoskeleton-based teleoperation frameworks such as GELLO [9] and ALOHA [10] control kinematic isomorphic versions of robotic arms, enabling accurate and direct tracking and mapping of the human wrist to the end of the robotic arm.

2.2 Visual Imitation Learning

Visual imitation learning seeks to map raw observations (e.g., RGB, depth, or point clouds) to actions via supervised learning. While classical behavior cloning suffers from compounding errors and limited generalization [11], recent methods reformulate policy learning as conditional generative modeling. Diffusion Policy [12] introduced the use of denoising diffusion models for action generation, enabling better handling of multimodal action distributions and temporal consistency. 3D Diffusion Policy [1] extends this idea by conditioning on 3D point clouds, improving generalization across space and appearance. RDT-1B [2] further scales diffusion-based policies to foundation models using cross-robot data and a unified action space, enabling strong zero-shot and few-shot performance.

Other relevant approaches include implicit behavior cloning (IBC) [11], trajectory optimization with diffusion models [13], and grasp generation via diffusion sampling [14]. Compared to these, policy-level diffusion models like DP3 and RDT-1B offer direct action synthesis for closed-loop control, with improved robustness, scalability, and data efficiency.

3 Method

Given a dataset of expert demonstrations $\mathcal{D} = \{(o_t, a_t)\}$, where o_t are visual observations and a_t are corresponding robot actions, our goal is to learn a policy $\pi(a_t|o_t)$ capable of reproducing complex manipulation behaviors. We study three methods in increasing order of complexity and generality.

3.1 Diffusion Policy (DP)

DP formulates policy learning as a denoising diffusion process over action sequences. Starting from a noisy action vector $a_t^{(k)}$, the model predicts the original clean action a_t by learning a noise prediction network ϵ_θ. The training objective minimizes the mean squared error between predicted and true noise, conditioned on the current observation:

$$L = \mathbb{E}_{t,k,\epsilon}\left[\|\epsilon - \epsilon_\theta(o_t, a_t^{(k)}, k)\|^2\right] \tag{1}$$

DP has several advantages: it models multimodal distributions effectively, scales well to high-dimensional action spaces, and exhibits stable training dynamics.

3.2 3D Diffusion Policy (DP3)

DP3 extends DP by incorporating 3D visual representations, particularly sparse point clouds derived from RGB-D data. The core innovation is a lightweight point cloud encoder that maps cropped and downsampled 3D points into a compact latent representation v. The diffusion policy then conditions on this 3D representation along with the robot pose to generate actions:

$$a_t^{(k-1)} = \alpha_k\left(a_t^{(k)} - \gamma_k \epsilon_\theta(a_t^{(k)}, v, q_t, k)\right) + \sigma_k \mathcal{N}(0, I) \tag{2}$$

DP3 shows improved generalization to variations in viewpoint, object instances, and spatial configurations, outperforming 2D visual policies with fewer demonstrations and lower safety violations.

3.3 Robotic Diffusion Transformer (RDT)

Language Embedding Due to GPU memory limitations (e.g., RTX 4090 with <24GB VRAM), we precomputed instruction embeddings using the T5-v1.1-XXL language model [15]. This was done using the official `encode_lang_batch.py` script, and the resulting embeddings were saved and loaded during both training and inference.

Model Architecture and Fine-Tuning. We adopt the RDT-1B [2] as our policy backbone for learning from demonstration data. The overall architecture is illustrated in the original paper. In brief, RDT consists of a diffusion transformer model conditioned on multi-modal inputs, including low-dimensional proprioceptive states, robot actions, language instructions, and multi-view visual observations from external and wrist-mounted cameras.

Visual tokens are extracted using a frozen SigLIP [16] encoder, while the instruction tokens are encoded using a frozen T5-XXL language model [15]. These pre-trained encoders are not updated during fine-tuning. Instead, we optimize the parameters of the diffusion transformer and associated MLP layers that process the input embeddings. The model is trained to denoise future actions over a action chunk, based on both the current robot state and task context.

At each diffusion timestep k, the ground-truth action $\mathbf{a}_t$ is corrupted according to

$$\mathbf{a}_t^{(k)} = \sqrt{\bar{\alpha}_k}\,\mathbf{a}_t + \sqrt{1-\bar{\alpha}_k}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I), \tag{3}$$

and the noisy action $\mathbf{a}_t^{(k)}$, together with the instruction ℓ, observation $\mathbf{o}_t$, and timestep k, is passed through our network:

$$\widehat{\mathbf{a}}_t = f_\theta\big(\ell,\, \mathbf{o}_t,\, \mathbf{a}_t^{(k)},\, k\big). \tag{4}$$

We optimize the parameters θ by minimizing the mean-squared error loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,k,\boldsymbol{\epsilon}}\big[\,\|\mathbf{a}_t - f_\theta(\ell, \mathbf{o}_t, \mathbf{a}_t^{(k)}, k)\|^2\big]. \tag{5}$$

4 Experiments (Dataset, implementation, results)

4.1 Airbot500 Dataset

Hardware Setup. We collect the dataset and conduct our experiments using a single-arm teleoperation setup consisting of an Airbot Play robot arm and a kinematically isomorphic teaching arm, as illustrated in Figure 1. The Airbot Play is a 6-DoF manipulator equipped with a parallel gripper for performing object manipulation tasks. The teaching arm, which mirrors the kinematic structure of the Airbot Play, is used for kinesthetic demonstrations during data collection. Two Intel RealSense cameras are used for visual input: a RealSense D455, mounted externally, provides RGB-D data from a fixed third-person perspective; and a RealSense D435, mounted on the robot's wrist, captures RGB images from the wrist viewpoint. This configuration enables both wide-field observation of the workspace and fine-grained feedback during manipulation, supporting data collection and downstream tasks.

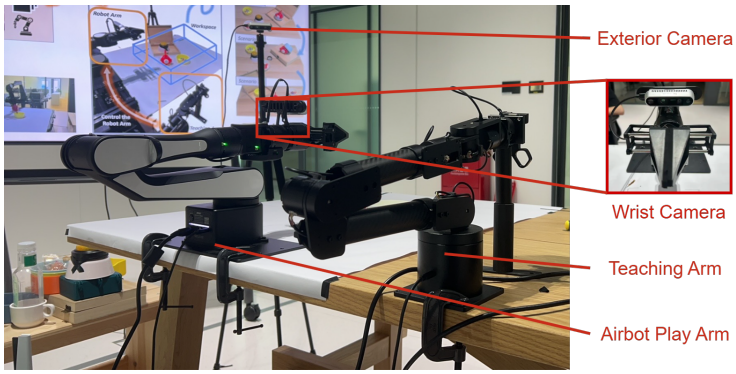


Figure 1: Hardware setup for our teleoperation system.

Data Statics. The Airbot500 dataset consists of 490 episodes totaling five sets of tasks. Each episode is 20s or 35s long (long-horizon task) and is recorded at a frequency of 25 frames per second. Each frame contains the following data: (i) timestamps; (ii) states of six arm joints and one gripper joint of the teacher arm; (iii) states of six arm joints and one gripper joint of the follower arm; (iv) RGB image and depth image from the external camera, both of which have a size of 480 × 640; and (v) RGB image from the wrist camera, which has a size of 480 × 640.

Tasks Description. In order to enrich the dataset and test the diffusion model from different perspectives, we set up the following five tasks:

(i) **catch_bird**: Pick up the red bird plush toy from the table and place it into the green square box / blue-gray hexagonal box. This task is a **simple grasping task**, which collects a total of 100 episodes. In order to enhance the generalization of the training, the scene sometimes has only one target box, and sometimes both boxes are present at the same time.

(ii) **catch_cup**: Pick up the cup from the table by its handle and place it onto the green square / blue-gray hexagonal platform. This task is a **functional grasping** task with a total of 100 episodes. To enhance the generalization of the training, the scene sometimes has only one target platform and sometimes another platform is placed at the same time.

(iii) **push_mouse**: Push the mouse on the table along a straight line backward / forward / left / right by a small distance (about 10-20 cm), and then click the left mouse button. This task is to test the model's ability to **understand planar position**, and a total of 60 episodes are collected.

(iv) **pour_water**: Pick up the transparent plastic water bottle from the table, pour some water into the cup (on the table or blue-gray hexagonal platform), and then place the water bottle back on the table. This task is to test the model's dexterity as well as its **spatial height comprehension**, and a total of 70 episodes are collected.

(v) **pack_duck**: Open the green box on the table using the handle on the lid, place the lid on the table, grab the blue / pink rubber duck on the table and put it into the box, then cover the box with the lid again. This task can be split into a total of six actions, grabbing the lid, placing the lid, grabbing the duck, placing the duck, picking up the lid, and closing the lid, as shown in Figure 2. Therefore it is a **long-horizon task** with a total of 160 episodes captured. In order to enhance generalization, the scenarios sometimes there is only one target duck, and sometimes there is another duck for obfuscation.

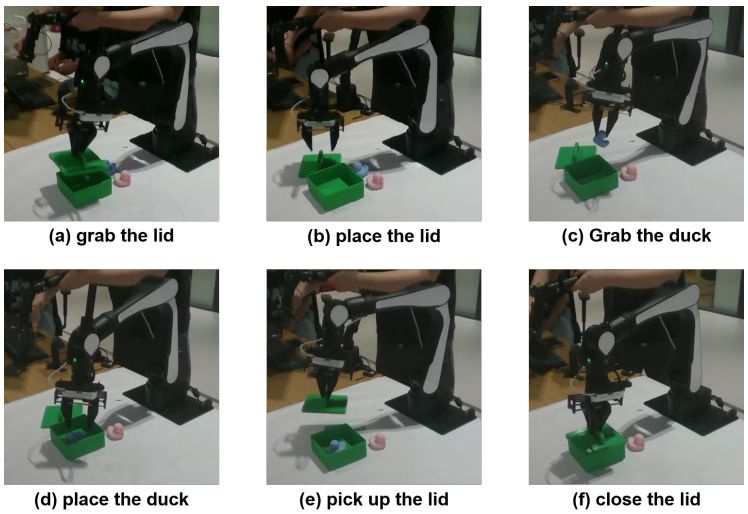


Figure 2: Action disassembly for the long-horizon task (pack_duck).

4.2 Experiments on 3D Diffusion Policy

The point cloud processing pipeline is a critical component of 3D Diffusion Policy (DP3), as the quality of 3D visual representations directly affects policy performance. In our implementation, we carefully design the workspace filtering and point cloud encoding stages to ensure both spatial relevance and visual precision of the input data.

3D Cube-Based Filtering and Dense Point Cloud Generation Initially, we attempted to define the workspace using a 2D bounding box in the RGB image space. While this method is simple to

implement, we found it insufficient in practice due to its lack of depth awareness. As shown in Figure 3 (top row), the resulting point cloud after 2D projection contained significant background noise and failed to capture task-relevant regions of the workspace, such as the manipulation surface and objects of interest. This illustrates the limitation of using 2D masks for 3D tasks—especially in scenes with significant depth variation or occlusions.

To improve precision, we transitioned to a 3D cube-based filtering approach. Specifically, we first transform depth images into 3D point clouds using the camera's intrinsic matrix. The dense point cloud is then cropped using an explicitly defined 3D cube bounding box, manually tuned to cover only the relevant manipulation area. This method allows us to isolate task-relevant geometry and eliminate background clutter. The bounds of the cube are iteratively adjusted by visualizing the cropped point cloud to ensure that all objects of interest are well-covered (Figure 3, bottom row).

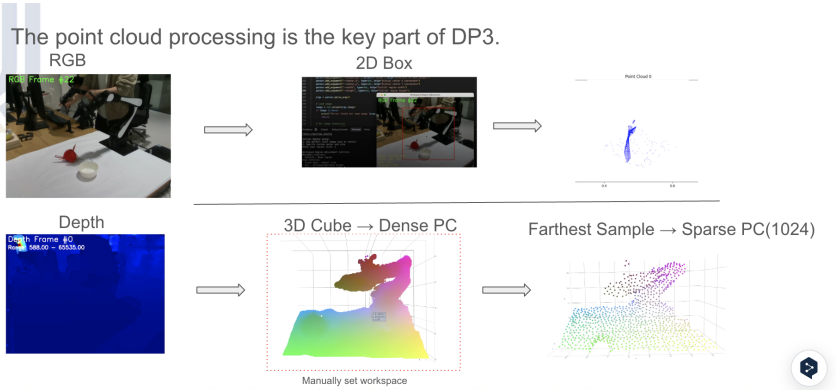


Figure 3: Point Cloud Extract Pipeline

After extracting the dense point cloud from the defined workspace, we apply Farthest Point Sampling (FPS) to downsample the input to a fixed size of 1024 points. This ensures computational efficiency and consistent input dimensionality while maintaining spatial diversity across the scene. Compared to uniform or random sampling, FPS better preserves the structural integrity of the scene and enhances downstream feature encoding. This step aligns with the original DP3 architecture, where a lightweight MLP-based point encoder processes the sparse 1024-point cloud to produce a compact visual embedding.

Data Preprocessing and Format Considerations Prior to training, the raw data had to be converted into the format required by DP3. This step proved to be non-trivial and time-consuming. For the 30-trajectory dataset, preprocessing and conversion took approximately 30 minutes. In this case, point cloud data were stored in .ply format. For the 100-trajectory dataset, which was stored using .npz files and accompanying depth image sequences, the conversion process took nearly five hours. Through this experience, we observed that the choice of data storage format significantly impacts preprocessing efficiency. Ultimately, we adopted the zarr format recommended by the DP3 implementation, which offers superior performance for large-scale dataset storage and access.

Training and Test Experiment We conducted our experiments on a simple task-specific setting, focusing primarily on a single-grasp manipulation task. This choice was motivated by time constraints and the need to validate the core functionality of 3D Diffusion Policy (DP3) in a controlled scenario. Due to the simplicity of the task, we did not extend our experiments to more complex behaviors or multi-stage interactions.

To investigate the effect of data scale, we trained two separate models using different datasets. The first dataset contained 30 grasp demonstrations, while the second consisted of 100 demonstrations. All training was performed on an NVIDIA RTX 3090 GPU.

In terms of training time, the 30-sample model was trained for just over two hours before early stopping, while the 100-sample model was trained for approximately four hours. However, we encountered several performance bottlenecks. The model trained on 30 demonstrations exhibited highly limited behavior: the robot arm moved in a single direction and failed to actuate the gripper.

With 100 demonstrations, the behavior slightly improved—sometimes the robot adjusted its arm pose—but it still struggled to localize the object precisely and could not control the gripper effectively.

Result Analysis Although we followed DP3's official training protocols and performed careful preprocessing, the learned policies showed limited robustness. We attribute this to the small dataset size, low diversity, and the inherent limitations of task-specific models like DP3. The model was sensitive to background clutter and object position shifts. To reduce such variance, future data collection will adopt a cleaner, more standardized setup.

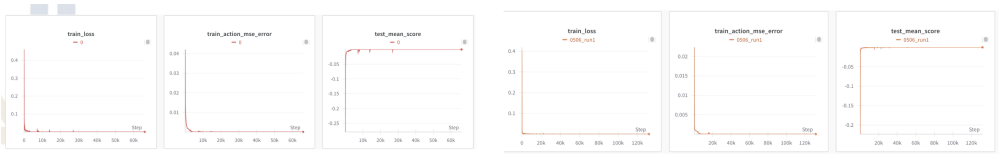


Figure 4: Training curves of DP3 on 30 (top row) and 100 (bottom row) demonstrations. Left to right: train loss, train action MSE, and test mean score.

The training curves (Figure 4) also suggest issues with learning dynamics. Both training loss and action MSE drop quickly—converging in under 10k steps—but the test mean score remains near zero, indicating poor generalization. This premature convergence suggests the model may have learned only superficial patterns.

The cause of this phenomenon is still unclear. It could stem from dataset limitations, model implementation issues, or insufficient 3D input quality. Further debugging and ablation studies are needed to pinpoint the underlying reasons.

4.3 Experiments on Fine-tuned RDT-1B

Before discussing experimental results, we first emphasize a key limitation of our setup: despite collecting tens of thousands of data points (e.g., 500 episodes, each lasting 30 seconds at a control frequency of 25 Hz), the dataset remains **small** compared to those used in large-scale image generation or natural language processing tasks. As a result, the model is **prone to overfitting** and far from a "foundation model." As noted by Benjamin [17] in a recent talk (Figure 5), collecting high-quality demonstrations for robot learning is never easy.

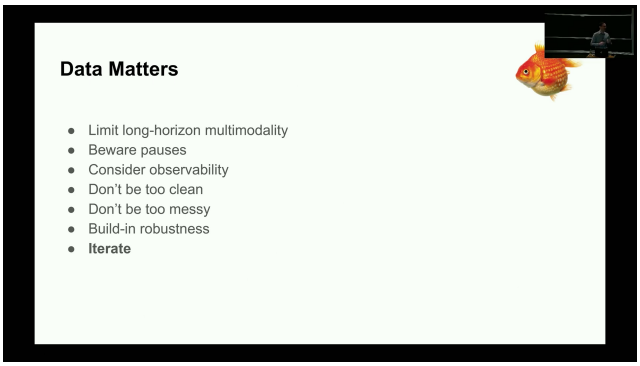


Figure 5: Data matters.

Implementation Details. The diffusion model was fine-tuned for 150,000 optimization steps on a single NVIDIA A100 GPU, which takes approximately six days on the university's HPC cluster. We employed a constant learning rate of 1×10^{-4} and trained on 490 episodes from the Airbot500 dataset at a control frequency of 25 Hz. We used a training batch size of 32. Sampling-based evaluation was performed every 500 steps using a sample batch size of 64. Training curves (see Figure 6) show a stable learning rate schedule, a smoothly decreasing total loss, and steadily declining reconstruction error, confirming convergence.

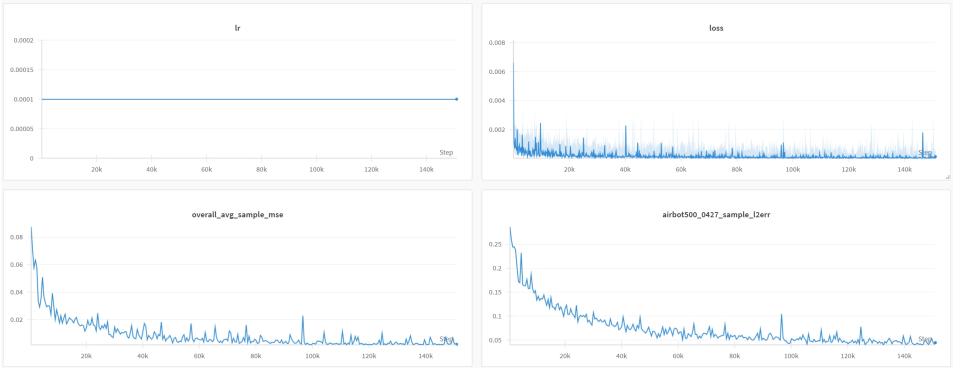


Figure 6: Training curves for RDT-1B fine-tuning process. Top Left: The learning rate is held constant at 1×10^{-4} throughout all 150,000 steps. Top Right: The total loss function decreases rapidly in the early phase and then converges smoothly. Bottom Left: The overall average sample reconstruction MSE, averaged over all episodes, declines steadily, indicating improved denoising accuracy. Bottom Right: The L2 reconstruction error on a representative Airbot500 episode falls continuously as training proceeds, demonstrating per-episode performance gains.

Real-Robot Results on Seen Tasks. Table 1 shows real-robot evaluation results on seen tasks from the Airbot500 dataset. The model achieves its highest success rate of 0.72 on the *catch_bird* task under the *single_box* condition, completing 18 trials out of 25 with an average execution time of 26.47 seconds. As the task setting becomes more complex, performance deteriorates; for example, the success rate drops to 0.35 under the *two_boxes* configuration, highlighting the increased difficulty of perceiving and reasoning over more cluttered scenes.

Due to hardware safety concerns, the number of trials for the *catch_cup* task was limited. Specifically, the robot gripper might crash into table-mounted platforms. We terminate this task after 10 trials. Further task design considerations are discussed in Appendix A.2.

The *pack_duck* task represents a long-horizon challenge, with a time limit twice as long as other tasks and a more complex sequence of actions. The policy achieves success rates of 0.69 and 0.62 in the initial stages—*open_lid* and *put_lid*, respectively—but always fails to complete the third stage, *pick_duck*, due to imprecise grasping. Despite this, we observe that the model is capable of making reactive decisions based on real-time observations rather than merely replaying memorized action sequences. Details of this validation can be found in Appendix A.3.

The policy fails to complete any trials of the *pour_water* and *push_mouse* tasks. We attribute this to the lack of explicit state transitions in the collected training episodes, which likely impedes the model's ability to infer appropriate actions. The limitations of our task design are further elaborated in Appendix A.2.

Overall, these results indicate that while the model demonstrates strong performance in structured and moderately complex scenarios, it struggles to generalize to tasks with clustered scene, longer horizons, or poorly defined state transitions.

Table 1: Real-Robot Results on Seen Tasks in The Airbot500 Datasets

Task	Type	Success Rate	# Trials	Finish Time (s)
catch_bird (20s)	single_box	0.72	25	26.47
	two_boxes	0.35	20	24.66
catch_cup (20s)	green_platform	0.20	5	30.00
	gray_platform	0	5	\
pack_duck (35s)	open_lid	0.69	13	\
	put_lid	0.62	13	\
	all	0	13	\
pour_water	\	(fail)	5	\
push_mouse	\	(fail)	5	\

Finish Time. As shown in Table 1 and Table 2, the robot generally completes tasks more slowly than the corresponding human demonstrations. This observation aligns with findings in prior work [18]. In our case, the execution speed is also influenced by the *max_step* parameter, which we set to *[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.3]*. Each value specifies the upper limit of the absolute angular change for a single joint in one step. The last value corresponds to the gripper, where the gripper state is defined within the range *[0, 1]*. Given that the control frequency is fixed at 25 Hz—consistent with the training data—predicted actions exceeding *max_step* are interpolated across multiple control cycles. This step-wise decomposition introduces additional delay, contributing to longer finish times. The effect of varying *max_step* on finish time is not further explored here.

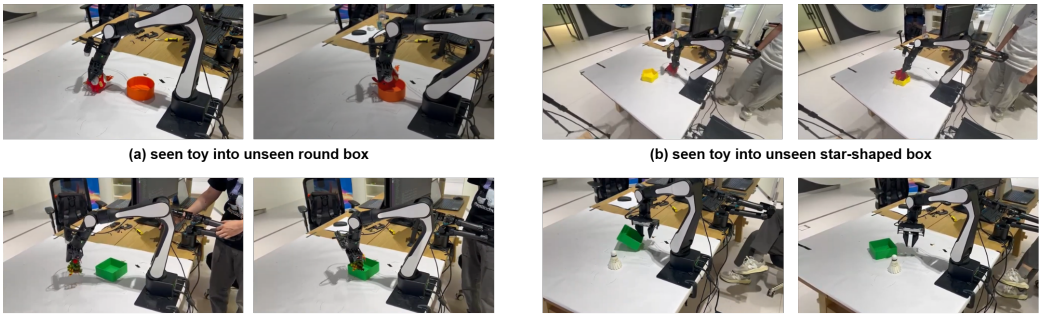
Generalizability Test. We evaluate two types of generalization, as illustrated in Figure 7. Table 2 shows the model's performance across these scenarios. For the seen object task *catch_bird*, the model achieves a high success rate of 0.72 when placing into a single seen container. However, the success rate drops to 0.30 when placing into an unseen container, indicating limited generalization to novel containers. For unseen object tasks, performance degrades further, with success rates of 0.10 for an unseen toy and 0.00 for other unfamiliar objects, reflecting significant difficulty in grasping object generalization. Finish times are relatively consistent across tasks. Some qualitative results are shown as Figure 8. Overall, the model generalizes moderately to unseen containers but performs poorly when handling completely novel objects.

Table 2: Comparison on Seen Objects and Unseen Objects

Task	Type	Success Rate	# Trials	Finish Time (s)
Catch_bird (20s)	single_box	0.72	25	26.47
	into_unseen_box	0.30	20	27.33
Unseen	unseen_toy	0.10	10	21.20
	other_unseen_obj	0	16	\

Figure 7: Examples of seen and unseen objects and containers used to evaluate generalization. The task involves grasping and placing objects into containers. Two types of generalization are considered: (1) **Container Generalization** — placing a seen object into unseen containers not present in the training set (top right), and (2) **Object Generalization** — placing unseen objects (bottom right) into seen containers.

Failure Case Study. we examine three representative task settings and analyze common failure cases using the Sankey diagrams in Figure 9. In the single-box setting (Figure 9a), most failures (4 out of 7) stem from grasping errors, with additional minor cases caused by placing mistakes, hesitation (where the robot arm hovers around the initial pose), and one timeout. When an additional distractor box is introduced (Figure 9b), grasping errors remain the dominant failure mode (9 out of 13), suggesting that the presence of multiple containers does not significantly confuse the placing step but increases the overall task complexity. In the unseen-box setting (Figure 9c), although grasping

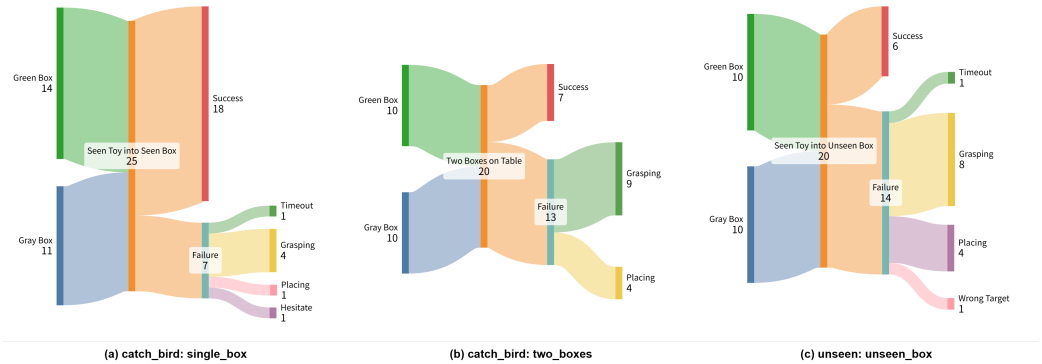


(a) seen toy into unseen round box (b) seen toy into unseen star-shaped box

(c) unseen toy into seen box (d) unseen shuttlecock into seen box

Figure 8: Qualitative results of generalization tests. (a–b) The model places a seen toy into novel containers with different shapes and colors. (c) The model successfully grasps an unseen Christmas tree toy and places it into a seen box. (d) The model fails to recognize the unseen shuttlecock, resulting in unsuccessful attempts.

errors still account for the largest portion of failures (8 out of 14), placing errors become more prominent (4 cases), and one failure is due to selecting the wrong grasp target. Additionally, we further discuss an overfitting issue in Appendix A.4.



(a) catch_bird: single_box (b) catch_bird: two_boxes (c) unseen: unseen_box

Figure 9: Sankey diagrams illustrating success and failure cases across different settings. (a) Single seen box: the model places a seen toy into a single familiar container. (b) Two-box setting: two seen boxes in the workspace. (c) Unseen-box setting: the model attempts to place a seen toy into a novel container. Failure types include grasping errors, placing errors, timeouts, hesitation, and wrong target selection.

5 Contribution

Table 3 shows the contributions of each group member to the project.

Table 3: Individual contributions to the group project

Name	Contribution
Guowei HUAI	Conducted real-world robot experiments using the DP3 framework, developed scripts for data conversion and point cloud extraction, and contributed to the DP3 section of the project report.
Jiahong CHEN	Collected Airbot500 dataset and conducted real-robot experiments together with Yiming, wrote dataset collection scripts, and wrote part of the dataset section of the project report.
Yiming Zhu	Collected Airbot500 dataset and conducted real-robot experiments together with Jiahong, wrote RDT-1B finetuning scripts, finetuned RDT-1B on HPC cluster, and wrote the relevant sections of the project report.

References

[1] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

[2] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[3] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] Aadhithya Iyer, Zhuoran Peng, Yinlong Dai, Irmak Guzey, Siddhant Haldar, Soumith Chintala, and Lerrel Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024.

[5] Runyu Ding, Yuzhe Qin, Jiyue Zhu, Chengzhe Jia, Shiqi Yang, Ruihan Yang, Xiaojuan Qi, and Xiaolong Wang. Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning. *arXiv preprint arXiv:2407.03162*, 2024.

[6] Yuzhe Qin, Yueh-Hua Wu, Shaowei Liu, Hanwen Jiang, Ruihan Yang, Yang Fu, and Xiaolong Wang. Dexmv: Imitation learning for dexterous manipulation from human videos. In *European Conference on Computer Vision*, pages 570–587. Springer, 2022.

[7] Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. *arXiv preprint arXiv:2307.04577*, 2023.

[8] Sridhar Pandian Arunachalam, Sneha Silwal, Ben Evans, and Lerrel Pinto. Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. In *2023 ieee international conference on robotics and automation (icra)*, pages 5954–5961. IEEE, 2023.

[9] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. *arXiv preprint arXiv:2309.13037*, 2023.

[10] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[11] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit Behavioral Cloning.

[12] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion.

[13] Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. Planning with Diffusion for Flexible Behavior Synthesis, December 2022. arXiv:2205.09991 [cs] TLDR: This paper considers what it would look like to fold as much of the trajectory optimization pipeline as possible into the modeling problem, such that sampling from the model and planning with it become nearly identical.

[14] Joao Carvalho, An T. Le, Philipp Jahr, Qiao Sun, Julen Urain, Dorothea Koert, and Jan Peters. Grasp Diffusion Network: Learning Grasp Generators from Partial Point Clouds with Diffusion Models in SO(3)xR3.

[15] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[16] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023.

[17] Benjamin Burchfiel. Diffusion for robotics. `https://www.youtube.com/watch?v=7tsCN2hRBMg`, 2025. Accessed: 2025-05-15.

[18] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[19] OpenAI. Gpt-4o technical report. `https://openai.com/index/gpt-4o`, 2024. Accessed: 2025-05-15.

A Appendix

In this Appendix, we provide additional insights into our work, including a list of the language instructions used in our experiments, a reflection on the task design, an analysis of RDT-1B's state perception performance, and an analysis of the overfitting issue.

A.1 Language Instructions Used in Experiments

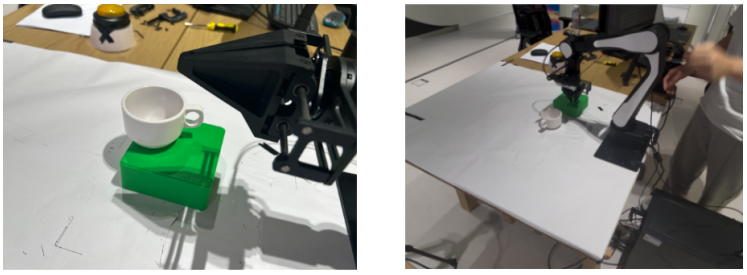
We list the language instructions used in the training dataset. Objects highlighted in olive represent those directly manipulated by the robot arm, while those in blue indicate containers involved in the task. Following the RDT-1B paper, we augment each instruction with both a simplified and a detailed version using ChatGPT-4o [19]. All instructions are embedded using the T5-v1.1-XXL model [15]. During the generalizability tests in Sec. 4.3, we keep the action verbs in the instructions fixed and vary only the names of the objects and containers.

Table 4: Task names and corresponding language instructions

Task Name	Language Instruction
`catch_bird`	Pick up the red bird plush toy from the table and place it into the blue-gray hexagonal box.
`catch_cup`	Pick up the cup from the table by its handle and place it onto the olive square platform.
`push_mouse`	Push the mouse on the table along a straight line forward by a small distance (about 10-20 cm), and then click the left mouse button.
`pack_duck`	Open the olive box on the table using the handle on the lid, place the lid on the table, grab the blue rubber duck on the table and put it into the box, then cover the box with the lid again.
`pour_water`	Pick up the transparent plastic water bottle from the table, pour some water into the cup, and then place the water bottle back on the table.

A.2 Reflection on Task Design

Experiment Safety. Certain tasks posed safety risks, leading to potentially dangerous robot behaviors such as collisions with the table or containers. To prevent equipment damage, we suspended these high-risk tasks after observing multiple failure cases during early trials. Figure 10 illustrates an example from the *catch_cup* task.



(a) Success (b) Crash

Figure 10: Example outcomes of *catch_cup* task. (a) Successful placement of a cup onto the platform. (b) Failure case where the robotic arm crashes due to inaccurate motion prediction or execution.

Explicit State Transition. RDT-1B heavily relies on the observed state during the inference stage, as it conditions the denoising process on image and textual embeddings. **To facilitate learning, the visual progression of the task should be made as distinct and informative as possible.** However, certain task designs in our benchmark, such as *pour_water* and *push_mouse*, lack explicit state transitions. Despite involving very different actions, these tasks exhibit highly similar or even visually identical frames at the beginning and end, as illustrated in Figure 11a and Figure 11b. Since RDT-1B

limits the image history to the two most recent frames, it struggles to differentiate between such visually ambiguous states, yet is expected to predict opposite actions (e.g., at $T = 0$ the robot begins the task; at $T = 499$ it returns to the same initial pose). Consequently, the model tends to regress to an averaged behavior that minimizes loss, leading the robot to remain at or near the initial pose, performing hesitant or unproductive movements without truly engaging in the task. This behavior is illustrated in Figure 11c.

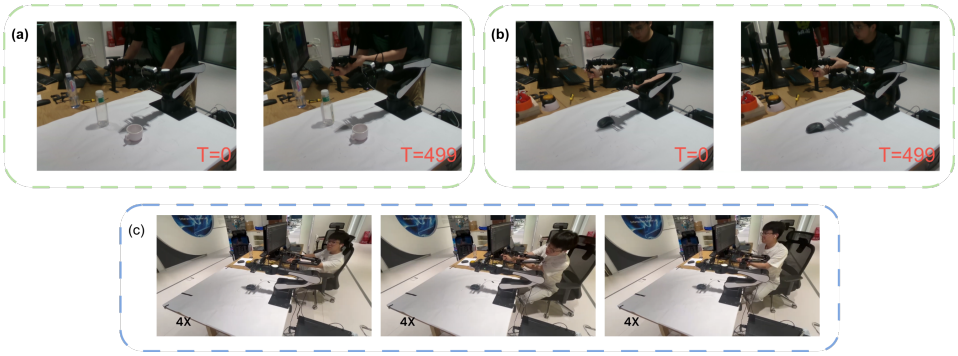


Figure 11: Visualization of ambiguous state transitions in tasks with visually similar start and end states. (a) and (b) show two tasks—*pour_water* and *push_mouse*—where the initial ($T = 0$) and final ($T = 499$) frames appear nearly identical, making it difficult for the model to infer meaningful transitions. (c) shows a failure case of *push_mouse*, where the robot remains near the initial pose.

A.3 Real-Time Decision Making of RDT-1B

Although the robot fails to complete the long-horizon task *pack_duck* due to inaccurate grasping, this experiment still demonstrates the RDT-1B model's ability to make prompt, state-dependent decisions rather than merely memorizing and replaying predefined action sequences such as "pick" and "place." As shown in Figure 12, when the human operator interrupts the task and manually places the rubber duck into the box—compensating for the robot's failed grasp—the model correctly detects the updated state. Instead of repeatedly attempting to grasp the duck, the model recognizes that the object is already in the box and proceeds to the next step: picking up the lid and closing the box. This behavior highlights the model's capacity for real-time, state-aware decision making, even when deviations from the expected action sequence occur.

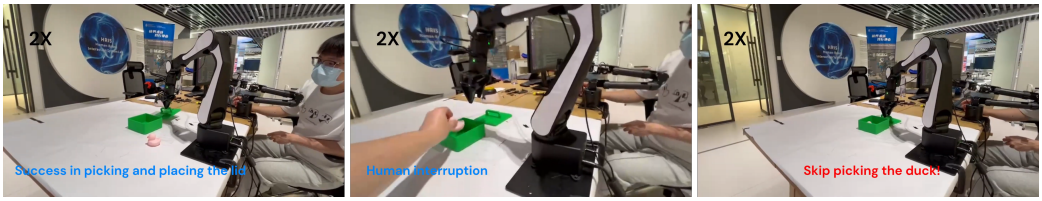


Figure 12: State-aware behavior of the RDT-1B model during the *pack_duck* task. Left: The robot successfully picks and places the lid. Middle: A human operator manually places the duck into the box to compensate for failed grasps. Right: The model correctly detects the updated state and skips the duck-picking step, proceeding directly to the next action—picking up the lid.

A.4 Overfitting on Collected Data

As discussed in the Introduction (Sec. 1), the model is prone to overfitting on the training dataset due to the limited number of episodes and the lack of data augmentation. Figure 13 provides clear evidence of this issue.

During data collection, we used an external camera to obtain a consistent, top-down view of the workspace, as shown in Figure 13. However, we observed that the model often over-relies on subtle visual patterns, including the presence of human operators. Specifically, during inference, the robot

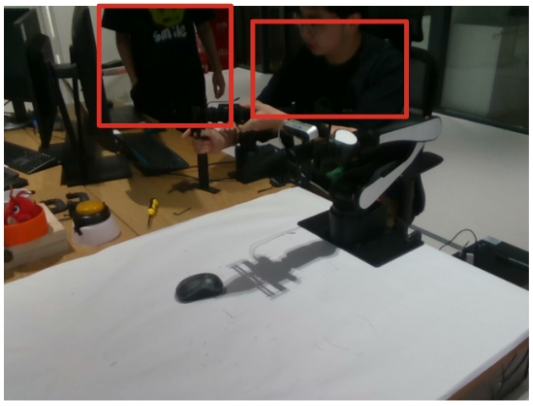


Figure 13: An example of human presence in the external camera view during data collection. The robot may overfit to such visual cues, leading to failure cases.

occasionally remains idle unless we "simulate" a teleoperation gesture—effectively prompting the controller to initiate movement. This suggests that the model has overfitted to human involvement cues present in the training data. Additionally, we found that maintaining a clean and static camera view is crucial. Any visual disturbances, such as clutter or unintended human presence in the frame, can negatively affect the model's performance. These observations further emphasize the importance of improving generalization through robust data collection and augmentation strategies.