RoboTwin: Dual-Arm Robot Benchmark with Generative Digital Twins (early version)

Yao Mu[*,1,3,†], Tianxing Chen[*,1,3,4], Shijia Peng[*,2,4], Zanxin Chen[*,2,4]
Zeyu Gao[5], Yude Zou[4], Lunkai Lin[2], Zhiqiang Xie[2], Ping Luo[1,†]
[1] The University of Hong Kong, [2] Agilex Robotics, [3] Shanghai AI Laboratory
[4] Shenzhen University, [5] Institute of Automation, Chinese Academy of Sciences
https://robotwin-benchmark.github.io/early-version

Abstract

In the rapidly advancing field of robotics, dual-arm coordination and complex object manipulation are essential capabilities for developing advanced autonomous systems. However, the scarcity of diverse, high-quality demonstration data and real-world-aligned evaluation benchmarks severely limits such development. To address this, we introduce RoboTwin, a generative digital twin framework that uses 3D generative foundation models and large language models to produce diverse expert datasets and provide a real-world-aligned evaluation platform for dual-arm robotic tasks. Specifically, RoboTwin creates varied digital twins of objects from single 2D images, generating realistic and interactive scenarios. It also introduces a spatial relation-aware code generation framework that combines object annotations with large language models to break down tasks, determine spatial constraints, and generate precise robotic movement code. Our framework offers a comprehensive benchmark with both simulated and real-world data, enabling standardized evaluation and better alignment between simulated training and real-world performance. We validated our approach using the open-source COBOT Magic Robot platform. Policies pre-trained on RoboTwin-generated data and fine-tuned with limited real-world samples improve the success rate of over 70% for single-arm tasks and over 40% for dual-arm tasks compared to models trained solely on real-world data. This significant improvement demonstrates RoboTwin's potential to enhance the development and evaluation of dual-arm robotic manipulation systems.

1. Introduction

Robotic systems with intricate dual-arm coordination and precise dexterity are essential for complex object manipulation to unlock advanced capabilities across domains such as healthcare, manufacturing, logistics, and domestic assistance. However, creating robust and versatile robotic systems that meet these demands remains a challenge, with a major bottleneck being the absence of diverse, high-quality training data and comprehensive evaluation benchmarks that are aligned with the real world.

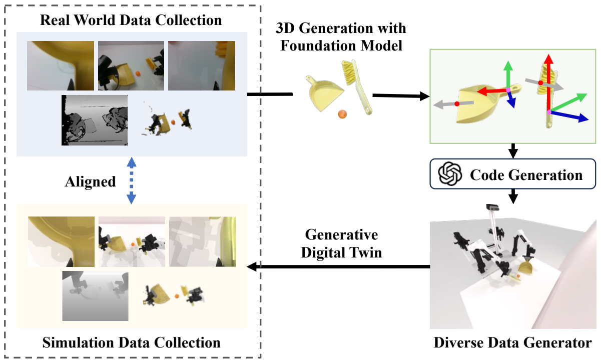
Real World Data Collection
Aligned
Simulation Data Collection
3D Generation with Foundation Model
Code Generation
Generative Digital Twin
Diverse Data Generator

Figure 1. **RoboTwin Benchmark.** A framework leveraging generative foundational models to generate realistic and interactive training scenarios and diverse expert demonstrations for benchmarking dual-arm robotic manipulation.

Traditional approaches to data collection, particularly human teleoperation [3, 10, 14, 16, 17, 28], yield high-quality demonstrations but face significant practical limitations. While these methods provide reliable training data, they are often prohibitively expensive, time-intensive, and struggle to cover the diverse range of scenarios robots encounter in real-world deployments. To address these limitations, researchers have turned to algorithmic trajectory generators in simulations [13, 20, 31]. These alternatives, however, frequently require task-specific design, hindering their generalizability and scalability. Recent advances such as MimicGen [47] and RoboCaca [51] have demonstrated significant progress in generating large-scale simulated expert data from limited human demonstrations. However, these

* Equal contribution. † Corresponding author.

approaches operate under fixed scenario settings and struggle to handle task variants beyond their predefined configurations, limiting their generalizability to novel scenarios.

Another limitation of existing benchmarks is that they predominantly focus on single-arm tasks [20, 48] or bimanual tasks with two separated arms [19], which fail to capture the complexity and coordination requirements inherent in integrated dual-arm systems. While HumanoidBench [54] and BiGym [11] explore benchmarks for humanoid bimanual manipulation, their scalability is limited by fixed environments or reliance on VR teleoperation for demonstration collection. As a result, these gaps highlight the urgent need for a scalable and standardized dual-arm collaboration benchmark with an efficient data collection pipeline.

To address these challenges, as shown in Fig. 1, we propose RoboTwin, a generative digital twin framework empowered by 3D generative foundation models and large language models (LLMs), aiming to produce diverse expert datasets and provide a real-world-aligned evaluation platform for dual-arm robotic tasks. Starting from a single 2D RGB image, we employ generative foundation models for 3D modeling and texture generation, enabling the efficient creation of varied object instances with different shapes, sizes, and appearances. Each object class is incorporated with spatial annotations, which define function axes, approach axes, lateral axes, and contact points and are applicable across various instances within an object class via feature point matching technology. Building upon these spatially-aware digital twins, RoboTwin leverages LLMs to interpret and decompose complex tasks into manageable sub-tasks. For each sub-task, we infer the constraints of the terminal state. For example, in a hammering task, the functional point of the hammer head needs to align with the surface of the target object. RoboTwin then generates executable code that calculates key poses based on these spatial constraints and object properties, interfacing with underlying planning modules to produce complete, feasible trajectories for execution.

Within the above framework, our RoboTwin features diverse dual-arm manipulation tasks that combine simulated expert data with real-world teleoperated datasets under consistent environmental and hardware setups. We then benchmark and evaluate the ability of RoboTwin to improve policy generalization in real-world scenarios. Experimental results demonstrated that policies pre-trained on 300 RoboTwin-generated samples and fine-tuned with 20 real-world samples improve the success rate by 70% in single-arm manipulation tasks like hammer beat, and over 40% in dual-arm coordination tasks, such as ball sweep, compared to those trained exclusively on 20 real-world samples.

We summarize our key contributions as: 1) we establish a convenient real-to-sim pipeline that requires only an RGB image from the real world to generate diverse 3D models of target objects, empowered by a 3D generative foundation model; 2) we create a spatial-aware code generation framework, which automatically creates expert-level demonstration data via a large language model and the spatial annotations of the target objects. 3) we develop a standard benchmark for dual-arm manipulation tasks including both real-world teleoperated data and high-fidelity synthetic data generated for corresponding scenarios. These advancements provide a robust framework for generating diverse, high-quality training data and policy evaluation for dual-arm manipulation tasks, significantly contributing to the development of more capable and versatile autonomous robotic systems.

2. Related Work

2.1. Datasets and Benchmarks for Robotics

To collect effective demonstrations for robotic tasks, human teleoperation is the most common approach, where human manually guides a robot across various tasks [16, 28, 43, 44, 46, 64]. Recent advancements have extended this methodology by employing teams of human operators over prolonged periods to assemble substantial real-world datasets [2, 6, 16, 28]. An alternative method employs algorithmic trajectory generators within simulators [13, 20, 27, 31, 63]. Nevertheless, such approaches typically demand manual, task-specific design for individual tasks. Recent initiatives like MimicGen [47] and RoboCaca [51] generate simulated expert data by adapting actions to new object poses, but remain limited to fixed scenarios and predefined task configurations. Furthermore, their reliance on fixed 3D objects limits the diversity of interacting objects and shapes.

In contrast, RoboTwin leverages 3D generative foundation models and LLMs to autonomously create both task variations and corresponding expert demonstrations. From 3D assets, it generates task scenarios and executable code via spatial reasoning, minimizing human intervention and supporting diverse object appearances.

2.2. Dual-arm Manipulation

While significant advances have been made in single-arm manipulation, coordinated multi-arm manipulation remains largely unexplored. Peract2 [19] offers benchmarks for bimanual tasks with separated arms, but its setup lacks the complexity of integrated dual-arm systems. HumanoidBench [54] evaluates dexterous, whole-body manipulation with a humanoid robot in a fixed reinforcement learning benchmark, while BiGym [11] provides a bimanual benchmark but is constrained by VR teleoperation, limiting their scalability in data collection and evaluation. As a benchmark for dual-arm tasks, RoboTwin enables automatic and large-scale coordinated manipulation data generation with comprehensive policy evaluation.

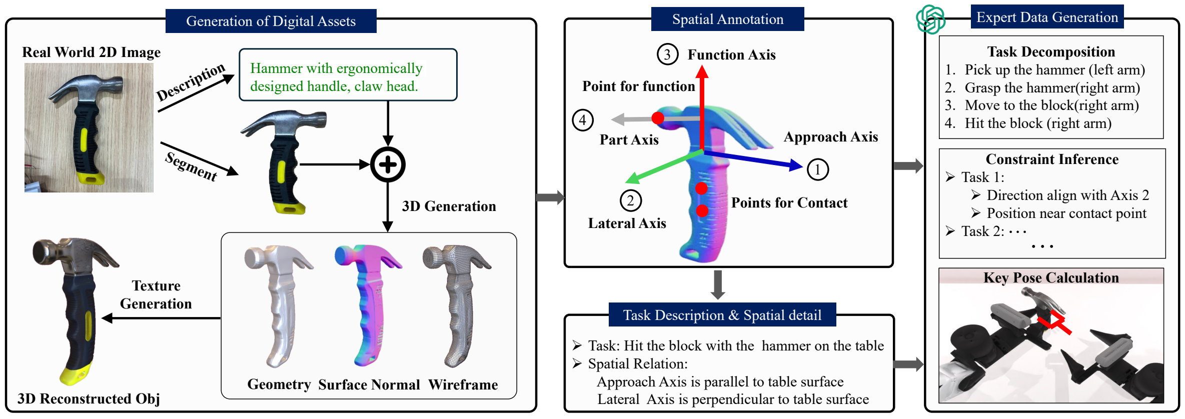
Generation of Digital Assets
Real World 2D Image
Description
Hammer with ergonomically designed handle, claw head.
Segment
3D Generation
Texture Generation
Geometry Surface Normal Wireframe
3D Reconstructed Obj
Spatial Annotation
③ Function Axis
Point for function
④ Part Axis
Approach Axis
①
② Lateral Axis
Points for Contact
Task Description & Spatial detail
Task: Hit the block with the hammer on the table
Spatial Relation:
Approach Axis is parallel to table surface
Lateral Axis is perpendicular to table surface
Expert Data Generation
Task Decomposition
1. Pick up the hammer (left arm)
2. Grasp the hammer(right arm)
3. Move to the block(right arm)
4. Hit the block (right arm)
Constraint Inference
Task 1:
Direction align with Axis 2
Position near contact point
Task 2: ···
Key Pose Calculation

Figure 2. **Real-to-simulation transfer and expert data generation.** We first leverage a 3D generative foundation model to create diverse 3D assets from 2D images, complete with geometry, normals, and textures. This process is augmented by vision-language models to generate variations of object descriptions, enabling the creation of visually diverse yet functionally consistent 3D models. We then implement a spatial annotation framework that marks key functional and contact points, along with functional, approach, and lateral axes on these 3D assets. Finally, we employ LLMs to generate expert demonstrations by decomposing tasks into subtasks, inferring spatial constraints, and generating collision-free robot behavior executable code that satisfies kinematic requirements.

2.3. Robot Manipulation Learning Methods

The adoption of human demonstrations to instruct robots in manipulation skills is a prevalent method in Robot Manipulation Learning [4, 5, 29, 42, 56]. Among the techniques, Behavioral Cloning stands out for learning policies offline from these demonstrations. It replicates observed actions from a curated dataset [6, 13, 16, 28, 31, 45, 53, 64**?**]. Conversely, Offline Reinforcement Learning enhances policy learning by optimizing actions based on a predefined reward function and exploiting large datasets [7, 21, 33–36]. The Action Chunking with Transformers (ACT) technique integrates a Transformer-based visuomotor policy with a conditional variational autoencoder to structure the learning of action sequences [57, 60, 65]. Diffusion models have been introduced into robot imitation learning and are gradually becoming a mainstream approach due to their excellent generative capabilities [30, 38–40, 52**?**]. Recently, the Diffusion Policy method has gained prominence. It employs a conditional denoising diffusion process for visuomotor policy representation, effectively reducing the accumulative error in trajectory generation that is often observed in Transformer-based visuomotor policies [12]. The 3D Diffusion Policy [62] uses point clouds for environmental observations, enhancing spatial information utilization and managing various robotic tasks in both simulated and real environments with only a small number of demonstrations.

2.4. LLM for Robotic Code Generation.

With their remarkable ability in natural language understanding and code generation, Large Language Models (LLMs) have revolutionized numerous domains in artificial intelligence. In robotics, these models have shown exceptional capabilities in bridging the gap between natural language commands and executable robot actions [8, 9, 15, 18, 22–26, 41, 50, 55]. Code as Policies [37] and RoboCodeX [49] established that LLMs can effectively translate high-level task descriptions into functional robot control programs. While Rekep [26] advances spatial reasoning between key points, it has limitations in handling functional axis constraints and fails to account for spatial relationships between object functional axes and the table surface during code generation. Furthermore, existing code generation approaches predominantly focus on single-arm robots, overlooking crucial aspects of dual-arm collaboration and active collision avoidance strategies.

3. Bridging Physical and Digital Worlds for Diverse Robot Behavior Generation

3.1. Generation of Diverse Digital Assets

Our approach utilizes Deemos's Rodin platform[*] to create 3D models from simple 2D RGB images. This method significantly reduces the need for expensive sensors while achieving realistic visual effects and supporting physical simulations. The process begins with capturing photographs of real-world objects. As shown in Fig. 2, we use GPT-4V [1] to analyze these images to generate corresponding descriptions, which are then autonomously modified via language model to create similar yet visually distinct object descriptions. We use these descriptions with Stable Diffusion XL Turbo to generate a diverse set of 2D images representing various appearances of the same object class. An image-conditioned 3D generation model then processes this collection of images, producing a wide range of 3D models for a single object type. The final output transforms a 2D image into a comprehensive 3D model, featuring detailed geometry, surface normals, wireframes, and textures. These elements not only enhance visual realism but also en-

[*]We use Deemos's 3D digital asset Generation Model (from text or image) Rodin: https://hyperhuman.deemos.com/rodin

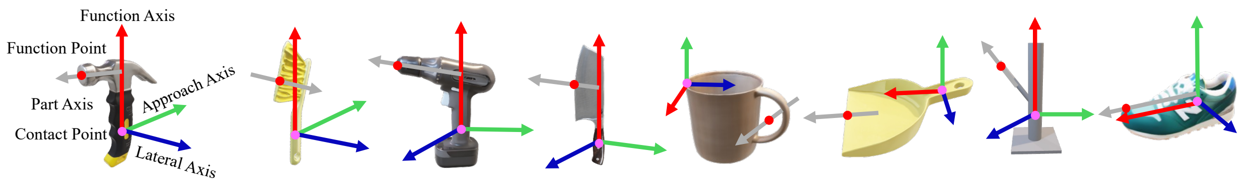

Figure 3. **Examples of spatial annotations.** Function and contact points with principal axes for functional parts and approach directions are extracted semi-automatically within RoboTwin for spatial- and geometry-aware manipulation and code generation.

sure compatibility with physics engines for advanced simulations.

3.2. Spatial Annotation Framework for 3D Assets

To enhance the structural integrity and universal applicability of generated assets, we implement a systematic approach for annotating key points and axes on tools. This methodology aims to render the data more comprehensible and accessible to large language models for complex task code generation. As shown in Fig. 3, the annotation process focuses on two primary elements: key points and axes.

Key Points. Key points represent specific locations on tools that are directly associated with their functional operations or user interaction points. We distinguish between these two types:

Point for Function: This key point designates the primary functional component of the tool, such as the striking surface of a hammer. It defines the tool's functional origin or point of action, directly correlating to the tool's primary purpose in a given task.

Point for Contact: This key point indicates the area of interaction between the tool and its user or other objects. It represents the gripping point or contact area, serving as a crucial human-machine interface point. Annotating this point facilitates understanding of tool's operational posture.

Axes. Axes are used to describe the spatial directionality of tools during task execution, encompassing the direction of functional execution and the tool's approach towards objects. We identify three principal axes:

Function Axis: This axis represents the direction in which the tool executes its primary function. It typically aligns with the tool's main operational vector, guiding the understanding of the tool's intended use and movement during task performance.

Approach Axis: The approach axis delineates the direction in which the tool approaches or is applied to the target object. This axis is crucial for comprehending the spatial relationship between the tool and its subject of operation.

Lateral Axis: This axis is perpendicular to both the function and approach axes, completing a three-dimensional coordinate system for the tool. The lateral axis aids in defining the tool's orientation and potential rotational movements during use.

By systematically annotating these key points and axes, we create a comprehensive spatial framework for each tool. This framework enables a more precise and context-aware understanding of tool functionalities, facilitating improved task planning and execution by large language models. We do not need to repeatedly annotate different 3D models from the same class. Instead, to streamline the annotation process for various 3D models of similar objects, we employ a feature point matching approach leveraging the Stable Diffusion encoder. This method enables the transfer of key points across various 3D models within the same object class. Our approach utilizes feature point matching to determine the target point. Specifically, under the table top view, given a source image I_s, a target image I_t, and a source point p_s, we aim to locate the corresponding point p_t in the target image. Following the methodology outlined in [32, 58], we extract diffusion features from both I_s and I_t. Since these diffusion features correspond to individual pixels in the target image, we can identify the pixel in I_t with the highest similarity to p_s by analyzing the extracted features. This technique allows for efficient key point migration across different 3D models of similar objects, eliminating the need for redundant annotations and enhancing the overall efficiency of the 3D modeling process.

3.3. Expert Data Generation

Building upon our spatial annotation framework and expert data generation pipeline, we present a systematic approach to generating robot behaviors that satisfy spatial constraints while ensuring collision-free execution. At the core of our framework lies a comprehensive dual-arm manipulation system with three key capabilities. First, it enables synchronized arm movements through screw motion interpolation coupled with coordinated gripper actions, ensuring stable object handling. Second, it supports independent arm operations for scenarios requiring asymmetric movements. Third, it implements dynamic collision avoidance through continuous adjustment of safe intermediate positions between arms. These foundational capabilities drive our behavior generation process, which integrates predefined APIs and leverages large language models (LLMs) to systematically generate expert data for robotic tasks. The process consists of the following steps:

1. **Scene Initialization**: The task environment is set up with relevant objects and their initial poses. For instance,

a hammering task would involve placing the hammer and target objects in their starting positions.

2. **Task Decomposition**: Based on human input describing the task, we use LLM to break it down into subtasks. For example, a "hammer a nail" task might be decomposed into: a) grasping the hammer, b) positioning the hammer over the nail, c) striking the nail, and d) returning the hammer to its original position.
3. **Constraint Inference**: For each sub-task, we use LLM to systematically infer spatial and temporal constraints through a hierarchical constraint analysis process. This analysis begins with identifying the functional relationships between objects' key points and axes. For grasping sub-tasks, we derive constraints between the end-effector's pose and the object's annotated contact points and approach axis, ensuring stable and effective grasps. For manipulation sub-tasks, we establish geometric constraints between the tool's functional points and the target object. These constraints encompass both positional alignments and directional requirements.
4. **Robot Behavior Generation**: Based on the derived spatial constraints, the LLM proceeds to generate corresponding behavioral code for each sub-task by calling relevant APIs (See prompts and examples in Appendix D). During execution, the system performs precise calculations of end-effector poses based on these spatial constraints. The process begins by identifying functional points on the object within the world coordinate system, which serves as the fundamental reference frame for all subsequent pose calculations. Building upon this foundation, our system implements a dual approach to determine optimal grasp poses. The first approach leverages pre-labeled contact points on the object to generate grasp poses. This method takes into account both the object's geometric properties and the robot's kinematic limitations. For more complex manipulation tasks, the second approach comes into play, computing grasp poses by aligning the object's functional point with a designated target point while adhering to specific directional constraints. To illustrate this, consider a hammering task: the system would align the hammer's head with the nail while calculating the proper orientation for an effective strike. This calculation factors in both the target approach direction (derived from the object's functional axes) and the desired orientation of the object once it's grasped. The core of behavior generation for each subtask is an optimization problem that seeks optimal joint trajectories $\theta(t)$. Using a screw motion planner, the system minimizes a cost function $J(\theta(t))$ while satisfying all task-specific constraints. This optimization is formulated as:

$$\min_{\theta(t)} \quad J(\theta(t))$$
$$\text{s.t.} \quad \begin{cases} \mathbf{T}_{\text{ee}} = f_{\text{FK}}(\theta(t)) & \text{(Kinematic constraint)} \\ \mathbf{P}_{\text{ee}} = \mathbf{P}_o - d \cdot \vec{a}_o & \text{(Position alignment)} \\ \vec{n}_{\text{ee}} = \vec{a}_o & \text{(Orientation alignment)} \\ \theta(t) \in \mathcal{C}, \forall t \in [t_0, t_f] & \text{(Collision avoidance)} \end{cases}$$

where, $J(\theta(t))$ represents a cost function that may incorporate factors such as energy efficiency, execution time, and motion smoothness. The constraints ensure that the robot's end-effector pose $\mathbf{T}_{\text{ee}}$ matches the desired pose calculated through the forward kinematics function $f_{\text{FK}}(\theta(t))$, aligning with the object's contact point $\mathbf{P}_o$ and approach axis $\vec{a}_o$ (position and orientation alignment). Finally, the trajectory $\theta(t)$ must remain within the collision-free configuration space $\mathcal{C}$ throughout the time interval $[t_0, t_f]$, ensuring collision avoidance. This comprehensive optimization framework enables the generation of robot behaviors that are efficient, satisfy spatial constraints, and guarantee safe, collision-free execution of complex tasks like hammering.
5. **Success Evaluation**: We implement criteria to assess successful task completion. For the hammering task, this might include verifying that the nail has been driven to the correct depth.
6. **Iterative Refinement**: The system gathers error data from multiple sources: runtime error messages, failed trajectory planning steps, and deviations between the final object states and their target configurations. To regenerate improved code, the system takes a comprehensive set of inputs including the collected error information, original task description, object annotations, and the previous version of code. The newly generated code is then tested, and if issues persist, the cycle continues until the desired performance is achieved.

4. Benchmark

Based on the methods introduced in Sec. 3, we design a comprehensive benchmark called RoboTwin to assess dual-arm robots, which includes 15 tasks in total. We employ the open-source Cobot Magic [†] platform as depicted in Fig. 4, which is equipped with four robot arms and four Intel RealSense D-435 RGBD cameras and is built on the Tracer chassis. These cameras are strategically positioned: one on the high part of the stand for an expansive field of view, two on the wrists of the robot's arms, and one on the low part of the stand which is optional for use. The front, left, and right cameras capture data simultaneously at a frequency of 30Hz. We utilize ManiSkill [59], an open-source simulation platform with GPU-accelerated data collection built on SAPIEN [61]. The details of each task in RoboTwin can be found in Appendix A.

[†]Platform Introduction: https://global.agilex.ai/products/cobot-magic

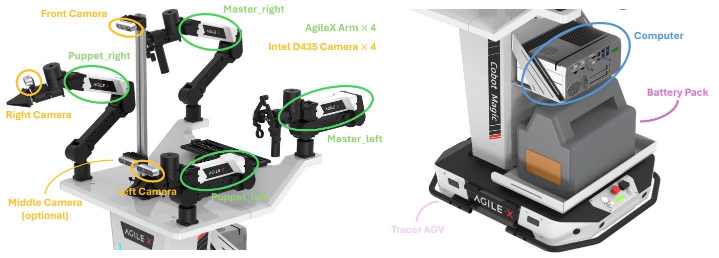

Figure 4. Illustration of our robot platform, with the capabilities for teleoperation and data acquisition.

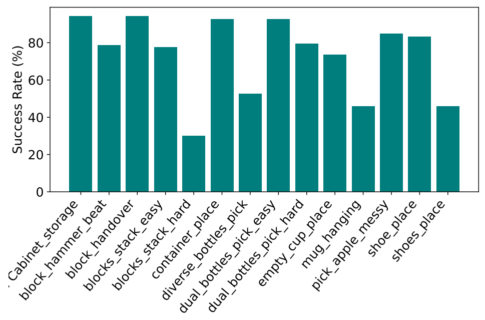

Figure 5. Success rate of the generated code for RoboTwin benchmark.

In RoboTwin benchmark, the agent needs to choose the appropriate collaboration method to successfully complete the task according to the distance of the target object from the left arm and the right arm. It involves the handover of the two arms, such as the handover task and putting the cup on the coaster, and the avoidance of interference between the two arms, such as the shoe placement task, which requires the two arms to coordinate with each other to place a pair of shoes in the limited space of the shoe box. The initial position and posture of the target objects in all our tasks are random. Before the scene is loaded, the mechanical dynamics accessibility of the randomly initialized scene will be checked to ensure that it is feasible. The task also includes objects of different shapes and appearances. The dual bottle pick task includes different models such as Coke bottles, Sprite bottles, and mineral water bottles, all of which are generated from 2D real pictures. The size of the objects in the environment is also randomized within a certain threshold. For each task, we provide well-designed script files that generate expert data across diverse scenarios, including various object placements and environmental conditions. We also report the success rate of generated code using our proposed method in Fig. 5, as described in Sec. 3.3.

For each task in our benchmark, we have pre-collected 100 sets of simulation data and 20 sets of real-world data. The hardware setup for the real-world experiments strictly matches that of the simulation environment. In both the simulation and real-world datasets, each captured frame consists of three images from the cameras, each providing an RGB and depth image at a resolution of 640×480 pixels. We also provide the point cloud data transformed from depth image, and colored point cloud data transformed from RGB and depth image for different types of algorithm evaluation. Additionally, the data includes the poses of the robotic arms' joints and end-effectors for both master and slave configurations, encompassing both left and right arms.

5. Experiment on RoboTwin Benchmark

5.1. Baselines

Diffusion Policy is a generative model for robotic imitation learning that models the distribution of potential actions to create diverse and complex action sequences. The approach has evolved into two main variants based on input dimensionality:

The 2D Diffusion Policy [12] processes two-dimensional visual information like images and video frames to predict actions for robotic manipulation tasks. While effective for many applications, this approach may have limitations in tasks requiring depth perception and spatial reasoning.

The 3D Diffusion Policy (DP3)[62] addresses these limitations by incorporating three-dimensional visual representations through point clouds. By using efficient point encoders to create compact 3D representations, DP3 enhances spatial awareness and demonstrates improved performance in tasks requiring complex spatial understanding.

5.2. Experimental Setup

We evaluated both 3D and 2D input imitation learning methods across 15 benchmark tasks, as shown in Fig. 6, tailoring our assessment approach to each model's characteristics. For the 3D Diffusion Policy (DP3), designed for few-shot learning, we tested performance in low-data regimes using 10, 20, and 50 expert demonstrations per task. This is aligned with DP3's network architecture, which is optimized for limited training data rather than large-scale datasets.

In contrast, we evaluated the 2D Diffusion Policy (DP), which processes image-based inputs, using larger datasets of 50 and 100 demonstrations per task. This approach accommodated DP's requirement for more substantial training data to develop robust policies, as we found that smaller datasets (10-20 demonstrations) proved insufficient for effective learning.

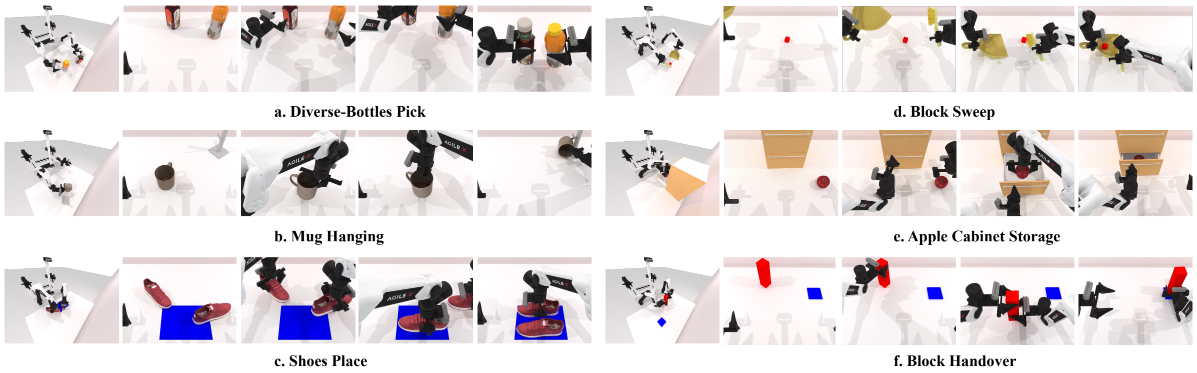

Figure 6. Examples of task execution in the RoboTwin benchmark.

	10	20	50		10	20	50
Apple Cabinet Storage				***Block Hammer Beat***			
DP3 (XYZ)	41%	59%	75%	DP3 (XYZ)	37%	45%	60%
DP3 (XYZ+RGB)	22%	41%	60%	DP3 (XYZ+RGB)	36%	41%	73%
Block Handover				***Block Sweep***			
DP3 (XYZ)	55%	89%	70%	DP3 (XYZ)	49%	80%	96%
DP3 (XYZ+RGB)	48%	81%	94%	DP3 (XYZ+RGB)	70%	98%	99%
Blocks Stack (Easy)				***Blocks Stack (Hard)***			
DP3 (XYZ)	/	/	/	DP3 (XYZ)	/	/	/
DP3 (XYZ+RGB)	0%	1%	23%	DP3 (XYZ+RGB)	0%	0%	3%
Container Place				***Diverse Bottles Pick***			
DP3 (XYZ)	34%	54%	68%	DP3 (XYZ)	3%	12%	38%
DP3 (XYZ+RGB)	18%	28%	54%	DP3 (XYZ+RGB)	0%	1%	7%
Dual Bottles Pick (Easy)				***Dual Bottles Pick (Hard)***			
DP3 (XYZ)	10%	48%	78%	DP3 (XYZ)	13%	29%	46%
DP3 (XYZ+RGB)	9%	41%	75%	DP3 (XYZ+RGB)	11%	26%	48%
Empty Cup Place				***Mug Hanging***			
DP3 (XYZ)	3%	30%	73%	DP3 (XYZ)	1%	9%	13%
DP3 (XYZ+RGB)	7%	23%	82%	DP3 (XYZ+RGB)	1%	2%	6%
Pick Apple Messy				***Shoe Place***			
DP3 (XYZ)	2%	2%	9%	DP3 (XYZ)	12%	16%	54%
DP3 (XYZ+RGB)	2%	3%	25%	DP3 (XYZ+RGB)	13%	20%	35%
Shoes Place				***Average***			
DP3 (XYZ)	2%	1%	12%	DP3 (XYZ)	20.15%	36.46%	53.23%
DP3 (XYZ+RGB)	0%	0%	5%	DP3 (XYZ+RGB)	17.93%	29.33%	45.93%

Table 1. Evaluation of 3D input diffusion policy. We tested the success rate with different numbers of demonstrations.

5.3. Experimental Results

5.3.1. Evaluation with 3D point cloud input

The experimental results are summarized in Table 1, which demonstrate a notable upward trend in success rates across various tasks as the training data increases. These results suggest a strong correlation between the number of expert demonstrations and task success, highlighting the effectiveness of the automatically generated expert data. The data further underscores the importance of ample training examples in the development of robust polices for complex tasks. Simultaneously, the findings reveal current limitations of imitation learning algorithms in bimanual tasks. In tasks requiring high levels of bimanual coordination, DP3 shows significant room for improvement, with relatively low success rates. For instance, the Block Stack task, which demands stacking scattered blocks in a specific color order, poses considerable challenges for the DP3 agent due to its requirements for bimanual coordination and object recognition. Similarly, the Shoes Place task, involving collaborative placement of a pair of shoes in a box, proves difficult for DP3. Notably, single-arm shoe placement tasks show success rates over five times higher, underscoring the complexity of bimanual operations. These findings highlight substantial research opportunities in imitation learning algorithms for highly coordinated bimanual tasks. Conversely,

	50	100		50	100
Apple Cabinet Storage	72%	64%	***Block Hammer Beat***	0%	8%
Block Handover	28%	100%	***Block Sweep***	95%	100%
Blocks Stack (Easy)	2%	8%	***Blocks Stack (Hard)***	0%	0%
Container Place	0%	20%	***Diverse Bottles Pick***	0%	20%
Dual Bottles Pick (Easy)	54%	94%	***Dual Bottles Pick (Hard)***	28%	62%
Empty Cup Place	20%	70%	***Mug Hanging***	0%	0%
Shoe Place	9%	16%	***Shoes Place***	0%	0%

Table 2. Evaluation of 2D input diffusion policy. We tested the success rate with different numbers of demonstrations.

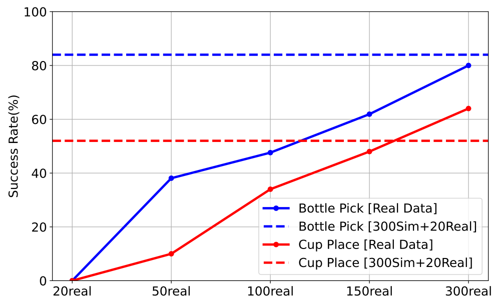

Figure 7. Comparison on scaling up real data and simulation data.

	Success Rate	
Task	**20 real**	**300Sim+20Real**
Bottle Pick (Easy)	0/50	**42/50**
Bottle Pick (Hard)	0/50	**16/50**
Container Place	0/50	**49/50**
Cup Place	1/50	**39/50**
Hammer Beat	2/50	**37/50**
Average	1.2%	**72%**

Table 3. Real world evaluation with a single arm.

tasks with more defined arm roles and similar motion requirements, such as Block Handover, Container Place, Dual Bottles Pick, and Cup Coasters Place, exhibit higher success rates for DP3, suggesting the efficacy of general imitation learning methods in these scenarios. The benchmark also includes challenging tasks in messy environments, where point cloud-only input methods struggle compared to those incorporating RGB data. Additionally, high-precision manipulation tasks like Mug Hanging further test the limits of robotic arm control, currently showing relatively low success rates. For thin objects that are challenging to process using point clouds alone, DP3 with RGB information outperforms its point cloud-only counterpart, as evidenced in tasks like Block Hammer Beat where visual recognition plays a crucial role.

5.3.2. Evaluation with 2D image input

In our experiments with 2D image inputs, as shown in Table 2, we observed that for most tasks, the success rate significantly improved as the amount of training data increased, with some tasks achieving up to 100% success rate. However, for more challenging dual-arm manipulation tasks, even with 100 training samples, the model failed to learn effective task completion strategies. In contrast, when using 3D inputs, we achieved measurable success rates for these complex tasks, indicating that 2D information alone is insufficient for complex manipulation. This finding highlights the pressing need for developing more efficient algorithms and scalable models that can effectively integrate both 2D and 3D features.

5.4. Real World Experiment

To validate the effectiveness of RoboTwin-generated training data in real-world policy deployment, we conducted comprehensive experiments on both single-arm and dual-arm manipulation tasks. We conducted a comparative experiment between policies trained solely on 20 real-world datasets and those pre-trained on 300 simulation datasets before fine-tuning on 20 real-world datasets (see more details and results in Appendix B).

The selection of 300 simulation datasets as our hyperparameter was based on empirical evidence shown in Fig. 7. Through progressive scaling of real-world data, we found that combining 300 simulation datasets with 20 real-world datasets yielded comparable performance than using 300 real-world datasets alone for both single-arm bottle pick and dual-arm cup placement tasks.

To investigate the performance disparity between baseline algorithms in single-arm versus dual-arm tasks, we conducted sim-to-real transfer experiments for both task categories. Each task underwent 50 test trials with randomized initial configurations, including varying object positions and orientations, as well as robot arm placements within predetermined boundaries. As shown in Table 3 and Table 4, experimental results revealed that policies trained on the combined dataset achieved markedly superior performance in real-world testing scenarios. Specifically, the integration of simulation data yielded a 72% improvement in success rates for single-arm tasks compared to policies trained exclusively on real-world data. For the more complex dual-arm tasks, we observed a significant improvement of over 40% in success rates. Our findings validate the effectiveness of our benchmark and data generation approach in bridging the sim-to-real gap, suggesting a promising direction for developing more robust and generalizable policies for dual-arm robotic manipulation tasks.

We observed significant disparities between single-arm and dual-arm scenarios. In the bottle rearrangement task, dual-arm operations presented substantially greater challenges, primarily due to the diverse initial states of target bottles (upright or lying down). While the incorporation of simulation data enabled the policy to achieve non-zero suc-

	Success Rate	
Task	**20 real**	**300Sim+20Real**
Dual bottle Pick (Easy)	0/50	**31/50**
Dual bottle Pick (Hard)	0/50	**11/50**
Container Place	25/50	**44/50**
Cup Place	0/50	**26/50**
Sweep Ball	25/50	**43/50**
Average	20%	**62%**

Table 4. Real world evaluation with dual arms.

cess rates, the overall performance remained suboptimal. This underscores the pressing need for developing more effective imitation learning algorithms specifically tailored to dual-arm coordination tasks.

6. Conclusion

This work introduces RoboTwin, a comprehensive benchmark integrating real-world and synthetic data for dual-arm robotic manipulation. Building upon the COBOT Magic Robot platform and leveraging 3D generative models for generative digital twins, our framework enables the efficient generation of diverse training data from single RGB images. Furthermore, our spatial-aware code generation framework automatically produces expert demonstrations by combining object annotations with LLMs to break down complex tasks and generate precise robotic movements. Through extensive experiments, we demonstrate that policies trained with RoboTwin-simulated data achieve significantly higher success rates with much less real data needed, compared to those trained solely on real-world data. These results validate our approach's effectiveness in bridging the sim-to-real gap while highlighting current limitations in dual-arm coordination tasks. Future works include developing more advanced algorithms specifically designed for dual-arm coordination and expanding the framework's capability to handle a broader range of complex manipulation tasks.

7. Acknowledgements

The authors extend their profound gratitude to D-robotics for their invaluable support in supplying the necessary cloud computing resources that facilitated the execution of this research. Furthermore, our sincere appreciation is extended to Deeoms for their contribution in providing essential model support, which was pivotal to the successful completion of this study.

References

[1] Gpt-4v(ision) system card. 2023. 3

[2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 2

[3] Jorge Aldaco, Travis Armstrong, Robert Baruch, Jeff Bingham, Sanky Chan, Kenneth Draper, Debidatta Dwibedi, Chelsea Finn, Pete Florence, Spencer Goodrich, et al. Aloha 2: An enhanced low-cost hardware for bimanual teleoperation. *arXiv preprint arXiv:2405.02292*, 2024. 1

[4] Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13778–13790, 2023. 3

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. 3

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022. 2, 3

[7] Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pages 3909–3928. PMLR, 2023. 3

[8] Guanyan Chen, Meiling Wang, Yao Mu Te Cui, Haoyang Lu, Tianxing Zhou, Zicai Peng, Mengxiao Hu, Haizhou Li, Yuan Li, Yi Yang, et al. Vlmimic: Vision language models are visual imitation learner for fine-grained actions. *arXiv preprint arXiv:2410.20927*, 2024. 3

[9] Junting Chen, Yao Mu, Qiaojun Yu, Tianming Wei, Silang Wu, Zhecheng Yuan, Zhixuan Liang, Chao Yang, Kaipeng Zhang, Wenqi Shao, et al. Roboscript: Code generation for free-form manipulation tasks across real and simulation. *arXiv preprint arXiv:2402.14623*, 2024. 3

[10] Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024. 1

[11] Nikita Chernyadev, Nicholas Backshall, Xiao Ma, Yunfan Lu, Younggyo Seo, and Stephen James. Bigym: A demo-driven mobile bi-manual manipulation benchmark. *arXiv preprint arXiv:2407.07788*, 2024. 2

[12] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023. 3, 6, 1, 4

[13] Murtaza Dalal, Ajay Mandlekar, Caelan Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. *arXiv preprint arXiv:2305.16309*, 2023. 1, 2, 3

[14] Runyu Ding, Yuzhe Qin, Jiyue Zhu, Chengzhe Jia, Shiqi Yang, Ruihan Yang, Xiaojuan Qi, and Xiaolong Wang. Bunny-visionpro: Real-time bimanual dexterous teleopera-

tion for imitation learning. *arXiv preprint arXiv:2407.03162*, 2024. 1

[15] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning*, pages 8469–8488. PMLR, 2023. 3

[16] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021. 1, 2, 3

[17] 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. 1

[18] Zeyu Gao, Yao Mu, Jinye Qu, Mengkang Hu, Lingyue Guo, Ping Luo, and Yanfeng Lu. Dag-plan: Generating directed acyclic dependency graphs for dual-arm cooperative planning. *arXiv preprint arXiv:2406.09953*, 2024. 3

[19] Markus Grotz, Mohit Shridhar, Tamim Asfour, and Dieter Fox. Peract2: Benchmarking and learning for robotic bimanual manipulation tasks, 2024. 2

[20] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659*, 2023. 1, 2

[21] Nico Gürtler, Sebastian Blaes, Pavel Kolev, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Bernhard Schölkopf, and Georg Martius. Benchmarking offline reinforcement learning on real-robot hardware. *arXiv preprint arXiv:2307.15690*, 2023. 3

[22] Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*, 2023. 3

[23] Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, 2023.

[24] Haoxu Huang, Fanqi Lin, Yingdong Hu, Shengjie Wang, and Yang Gao. Copa: General robotic manipulation through spatial constraints of parts with foundation models. *arXiv preprint arXiv:2403.08248*, 2024.

[25] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[26] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024. 3

[27] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. 2

[28] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, 2021. 1, 2, 3

[29] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022. 3

[30] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022. 3

[31] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. In *International Conference on Machine Learning*, 2023. 1, 2, 3

[32] Yuanchen Ju, Kaizhe Hu, Guowei Zhang, Gu Zhang, Mingrun Jiang, and Huazhe Xu. Robo-abc: Affordance generalization beyond categories via semantic correspondence for robot manipulation. *arXiv preprint arXiv:2401.07487*, 2024. 4

[33] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multitask robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021. 3

[34] Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. *arXiv preprint arXiv:2109.10813*, 2021.

[35] Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*, 2022.

[36] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020. 3

[37] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023. 3

[38] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. *arXiv preprint arXiv:2302.01877*, 2023. 3

[39] Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16467–16476, 2024.

[40] Zhixuan Liang, Yao Mu, Yixiao Wang, Fei Ni, Tianxing Chen, Wenqi Shao, Wei Zhan, Masayoshi Tomizuka, Ping Luo, and Mingyu Ding. Dexdiffuser: Interaction-aware diffusion planning for adaptive dexterous manipulation. *arXiv preprint arXiv:2411.18562*, 2024. 3

[41] Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. Moka: Open-vocabulary robotic manipulation through mark-based visual prompting. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024. 3

[42] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023. 3

[43] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018. 2

[44] Ajay Mandlekar, Jonathan Booher, Max Spero, Albert Tung, Anchit Gupta, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity. *arXiv preprint arXiv:1911.04052*, 2019. 2

[45] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. In *Robotics: Science and Systems (RSS)*, 2020. 3

[46] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Human-in-the-loop imitation learning using remote teleoperation, 2020. 2

[47] 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. *arXiv preprint arXiv:2310.17596*, 2023. 1, 2

[48] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3): 7327–7334, 2022. 2

[49] Yao Mu, Junting Chen, Qing-Long Zhang, Shoufa Chen, Qiaojun Yu, GE Chongjian, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. In *Forty-first International Conference on Machine Learning*, 2024. 3

[50] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[51] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024. 1, 2

[52] Fei Ni, Jianye Hao, Yao Mu, Yifu Yuan, Yan Zheng, Bin Wang, and Zhixuan Liang. Metadiffuser: Diffusion model as conditional planner for offline meta-rl. In *International Conference on Machine Learning*, pages 26087–26105. PMLR, 2023. 3

[53] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 3

[54] Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024. 2

[55] Hao Sha, Yao Mu, Yuxuan Jiang, Li Chen, Chenfeng Xu, Ping Luo, Shengbo Eben Li, Masayoshi Tomizuka, Wei Zhan, and Mingyu Ding. Languagempc: Large language models as decision makers for autonomous driving. *arXiv preprint arXiv:2310.03026*, 2023. 3

[56] Pratyusha Sharma, Lekha Mohan, Lerrel Pinto, and Abhinav Gupta. Multiple interactions made easy (mime): Large scale demonstrations data for imitation. In *Conference on robot learning*, pages 906–915. PMLR, 2018. 3

[57] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015. 3

[58] Luming Tang, Menglin Jia, Qianqian Wang, Cheng Perng Phoo, and Bharath Hariharan. Emergent correspondence from image diffusion. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 4

[59] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024. 5

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 3

[61] Fanbo Xiang, He Wang, Yuzhe Qin, Austin Wang, Hejia Zhang, Yikuan Xia, Binbin Lin, Yuzhe Wu, Chengcheng Tang, Yixin Zhu, Li Yi, Leonidas J. Guibas, and Hao Su. Sapien: A simulated part-based interactive environment. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 5

[62] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy. *arXiv preprint arXiv:2403.03954*, 2024. 3, 6, 1, 4

[63] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, 2020. 2

[64] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation

learning for complex manipulation tasks from virtual reality teleoperation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 2, 3

[65] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023. 3

RoboTwin: Dual-Arm Robot Benchmark with Generative Digital Twins (early version)

Supplementary Material

A. Task Description for RoboTwin

We provide detailed descriptions of all tasks involved in the benchmarks and real-world experiments, as shown in Table 5, totaling 15 tasks. The initial positions of target objects in all tasks are randomized. Some tasks must be completed using both arms, such as *Shoes Place*. Other tasks have both dual-arm and single-arm versions, like *Container Place* and *Empty Cup Place*. For these dual-arm versions, the appropriate arm is selected based on the object's initial position. Tasks like *Block Handover* and *Mug Hanging* involve handoffs between the left and right arms. More challenging tasks, such as *Shoes Place*, require high coordination between both arms.

B. Implementation Details for Simulation Experiments

B.1. Baseline Introduction and Setup

Diffusion Policy [12] is a novel approach in robot learning that models the robot's visuomotor policy as a conditional denoising diffusion process. It learns the gradient of the action-distribution score function and iteratively optimizes with respect to this gradient field during inference via a series of stochastic Langevin dynamics steps. This methodology enables the robot to generate diverse and high-dimensional action distributions, effectively handling multimodal behaviors and high-dimensional action spaces. The input to the Diffusion Policy is a sequence of visual observations, and the output is a sequence of actions predicted over a fixed duration, facilitating robust and temporally consistent action generation.

Building upon the Diffusion Policy, the 3D Diffusion Policy (DP3) [62] integrates 3D visual representations into the diffusion framework, enhancing the robot's ability to generalize across various tasks and environments. DP3 employs a compact 3D visual representation extracted from sparse point clouds using an efficient point encoder. The input to DP3 is a 3D scene representation, and the output is a sequence of 3D end-effector poses, including both translations and rotations, predicted over a fixed duration. This approach allows the robot to perform complex manipulation tasks with high precision and generalization capabilities, even with limited demonstrations.

We outline all the key hyper-parameters for DP [12] and DP3 [62] in Table 6. These hyper-parameters were adopted directly from the original DP and DP3 papers to ensure consistent performance and enable fair comparison with the published results.

For the camera settings, we utilize a 2D observation with an image resolution of (320, 240) and perform FPS downsampling on the point cloud obtained from the image to 1024 points for 3D observation.

C. Sim2Real Experiment Setup

Our real-world experiments aim to verify whether the generated simulation data can effectively aid in policy learning, enabling high performance in real-world testing despite exposure to only limited real-world data.

C.1. Simulation vs. Real Scene Visualization

We present the comparison images of the real and simulation for the same task in Fig. 8. The RoboTwin-generated data demonstrates exceptional visual fidelity to real-world scenarios across all tasks. The simulated environment achieves near photo-realistic quality, accurately capturing lighting, shadows, and object textures. This high-fidelity simulation shows great promise for robot learning by effectively bridging the sim-to-real gap.

C.2. Details of Sim2Real Fine-Tuning

To better align real-world and simulation images, and considering that brighter environments facilitate better policy learning and feature extraction, we enhanced the typically darker real-world observations. We applied the following brightness adjustment code, where the alpha parameter can be fine-tuned based on specific lighting conditions:

```
cv2.convertScaleAbs(src, alpha=1.5, beta=0)
```

Step 1: We pretrain a Diffusion Policy network using 300 sets of RoboTwin-generated simulation data. This simulation data provides a rich foundation for learning basic manipulation skills. The pretraining phase follows the hyperparameter settings detailed in Tab. 7.

Step 2: Following the pretraining phase, we implement a highly efficient fine-tuning approach using only 20 sets of real-world robot data. This minimal data requirement significantly reduces the burden of real-world data collection while still enabling effective domain adaptation. The fine-tuning process builds upon the pretrained policy network from Step 1, adjusting the network parameters to bridge the sim-to-real gap. All fine-tuning hyperparameters are carefully selected and documented in Tab. 7 to ensure optimal transfer learning performance.

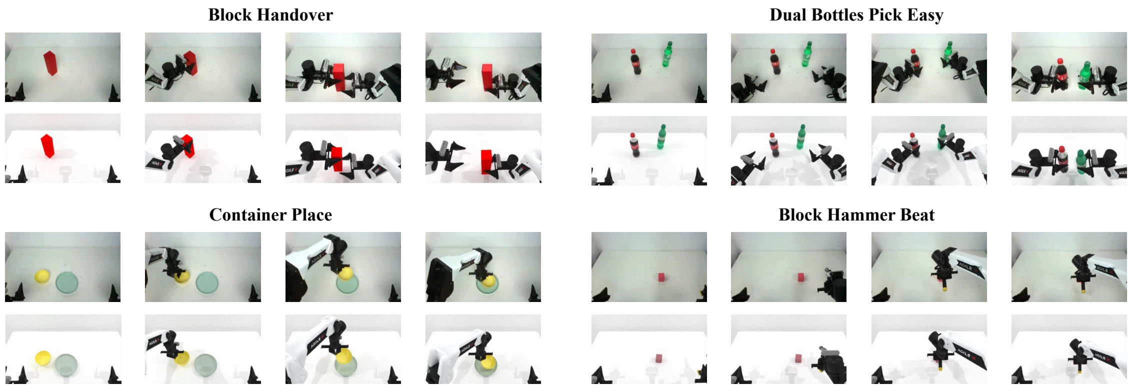

Figure 8. Visualization of real-world and RoboTwin-generated data. For each task, real-world collected data is shown in the top row, with RoboTwin-generated data displayed in the bottom row.

This two-stage training strategy effectively combines the advantages of abundant simulation data with minimal real-world data requirements, demonstrating an efficient approach to robot skill acquisition and transfer.

Task	**Description**
Block Hammer Beat	There is a hammer and a block in the middle of the table. If the block is closer to the left robotic arm, it uses the left arm to pick up the hammer and strike the block; otherwise, it does the opposite.
Block Handover	A long block is placed on the left side of the table. The left arm grasps the upper side of the block and then hands it over to the right arm, which places the block on the blue mat on the right side of the table.
Blocks Stack (Easy)	Red and black cubes are placed randomly on the table. The robotic arm stacks the cubes in order, placing the red cubes first, followed by the black cubes, in the designated target location.
Blocks Stack (Hard)	Red, green, and blue cubes are placed randomly on the table. The robotic arm stacks the cubes in order, placing the red cubes first, followed by the green and then the blue cubes, in the designated target location.
Bottle Adjust	A bottle is placed horizontally on the table. The bottle's design is random and does not repeat in the training and testing sets. When the bottle's head is facing left, pick up the bottle with the right robot arm so that the bottle's head is facing up; otherwise, do the opposite.
Container Place	Random containers (cups, bowls, etc.) are placed randomly on the table. The robotic arm moves the containers into a fixed plate.
Diverse Bottles Pick	A random bottle is placed on the left and right sides of the table. The bottles' designs are random and do not repeat in the training and testing sets. Both left and right arms are used to lift the two bottles to a designated location.
Dual Bottles Pick (Easy)	A red bottle is placed randomly on the left side, and a green bottle is placed randomly on the right side of the table. Both bottles are standing upright. The left and right arms are used simultaneously to lift the two bottles to a designated location.
Dual Bottles Pick (Hard)	A red bottle is placed randomly on the left side, and a green bottle is placed randomly on the right side of the table. The bottles' postures are random. Both left and right arms are used simultaneously to lift the two bottles to a designated location.
Dual Shoes Place	One shoe is placed randomly on the left and right sides of the table. The shoes are the same pair with random designs that do not repeat in the training and testing sets. Both left and right arms are used to pick up the shoes and place them in the blue area, with the shoe heads facing the left side of the table.
Empty Cup Place	An empty cup and a cup mat are placed randomly on the left or right side of the table. The robotic arm places the empty cup on the cup mat.
Mug Hanging (Easy)	A mug is placed randomly on the left side of the table, and a mug rack is placed on the right side (fixed). The left arm moves the mug to a suitable position in the middle of the table, and then the right arm hangs the handle of the mug on the mug rack.
Mug Hanging (Hard)	A mug is placed randomly on the left side of the table, and a mug rack is placed randomly on the right side. The left arm moves the mug to a suitable position in the middle of the table, and then the right arm hangs the handle of the mug on the mug rack.
Pick Apple Messy	Apples and four random items are placed randomly on the table. The robotic arm picks up the apple and lifts it.
Put Apple Cabinet	Initially, an apple is placed randomly. The robotic arm uses the left arm to open the cabinet and the right arm to pick up the apple and place them inside.
Shoe Place	Shoes are placed randomly on the table, with random designs that do not repeat in the training and testing sets. The robotic arm moves the shoes to a blue area in the center of the table, with the shoe head facing the left side of the table.
Tool Adjust	A tool is placed horizontally on the table. The tool's design is random and does not repeat in the training and testing sets. When the tool's head is facing left, pick up the tool with the right robot arm so that the tool's head is facing up; otherwise, do the opposite.

Table 5. Task descriptions for RoboTwin platform.

Parameter	DP [12]	DP3 [62]
horizon	8	8
n_obs_steps	3	3
n_action_steps	6	6
num_inference_steps	100	10
dataloader.batch_size	128	256
dataloader.num_workers	0	8
dataloader.shuffle	True	True
dataloader.pin_memory	True	True
dataloader.persistent_workers	False	False
optimizer._target_	torch.optim.AdamW	torch.optim.AdamW
optimizer.lr	1.0e-4	1.0e-4
optimizer.betas	[0.95, 0.999]	[0.95, 0.999]
optimizer.eps	1.0e-8	1.0e-8
optimizer.weight_decay	1.0e-6	1.0e-6
training.lr_scheduler	cosine	cosine
training.lr_warmup_steps	500	500
training.num_epochs	300	3000
training.gradient_accumulate_every	1	1
training.use_ema	True	True

Table 6. Hyper-parameter Settings for Training and Deployment of DP and DP3 Algorithms.

Parameter	Pre-training	Fine-tuning
horizon	8	8
n_obs_steps	3	3
n_action_steps	6	6
num_inference_steps	100	100
dataloader.batch_size	128	128
dataloader.num_workers	0	0
dataloader.shuffle	True	True
dataloader.pin_memory	True	True
dataloader.persistent_workers	False	False
optimizer._target_	torch.optim.AdamW	torch.optim.AdamW
optimizer.lr	1.0e-4	5e-5
optimizer.betas	[0.95, 0.999]	[0.95, 0.999]
optimizer.eps	1.0e-8	1.0e-8
optimizer.weight_decay	1.0e-6	1.0e-6
training.lr_scheduler	cosine	cosine
training.lr_warmup_steps	500	500
training.num_epochs	300	300
training.gradient_accumulate_every	1	1
training.use_ema	True	True
training.rollout_every	50	50

Table 7. Hyper-parameter Settings for Pretraining with RoboTwin-generated Data and Finetuning with Limited Real-world Data.

D. Prompts

In the process of generating expert demonstration data, we structure prompts for large language models with three components: 1) Task Information and General Prompt; 2) Introduction to Available APIs, detailing usable programming interfaces and libraries; 3) Function Examples that demonstrate implementation patterns.

D.1. Task Information and General Prompt

~~~
You need to generate relevant code for some robot tasks in a robot simulation environment based on the
 provided API.
In this environment, distance 1 indicates 1 meter long. Pose is representated as 7 dimention, [x, y, z,
 qw, qx, qy, qz]. For a 7-dimensional Pose object, you can use Pose.p to get the [x, y, z] coordinates and
 Pose.q to get the [qw, qx, qy, qz] quaternion orientation.
All functions which has parameter actor_data, and all of actor_data should be in the actor_data_dic.
In the world coordinate system, the positive directions of the xyz coordinate axes are right, front, and
 upper respectively, so the direction vectors on the right, front, and upper sides are [1,0,0], [0,1,0],
 [0,0,1] respectively. In the same way, we can get the unit vectors of the left side, back side and down
 side.

Task Discription:
Use the gripper to pick up block1 and move block 1 to the target position. Then pick up block 2 and place
 it on the block 1, and finally pick up block3 and place it on the block2. If block1's x coordinate (dim
 0) is greater than 0, use right arm to stack the block1, else use the left arm. And same for the block2
 and block3.
Note:
1. You need to call the get_avoid_collision_pose function to avoid collisions when the left and right
 arms move alternately.
2. For example, if the previous action uses the left arm and the next action uses the right arm, you need
 to move the left arm after release gripper to avoid collisions, vice versa.
3. The pre-dis of stacked blocks may be smaller.

Available Constants:
self.world_direction_dic: {
    'left':         [0.5,  0.5,  0.5,  0.5],
    'front_left':   [0.65334811, 0.27043713, 0.65334811, 0.27043713],
    'front' :       [0.707, 0,    0.707, 0],
    'front_right':  [0.65334811, -0.27043713,  0.65334811, -0.27043713],
    'right':        [0.5,    -0.5, 0.5,  0.5],
    'top_down':     [0,      0,   1,    0],
}
The world_direction_dic is a dict of different approach directions.
The Actor Name List: ['block1', 'block2', 'block3', 'block1_target_pose']
The Actor Data List: ['block1_data', 'block2_data', 'block3_data', 'block1_target_pose']

The Actor Points Discription: {
    'block1':{
        'contact_points':[]
        'target_points': ["The top surface center of the block." ],
        'functional_points': ["Point0: The center point on the bottom of the block, and functional axis
 is vertical bottom side down"]
        'actor_orientation': []
    },
    'block2':{
        'contact_points':[]
        'target_points': ["The top surface center of the block." ],
        'functional_points': ["Point0: The center point on the bottom of the block, and functional axis
 is vertical bottom side down"]
        'actor_orientation': []
    },
    'block3':{
        'contact_points':[]
        'target_points': ["The top surface center of the block." ],
        'functional_points': ["Point0: The center point on the bottom of the block, and functional axis
 is vertical bottom side down"]
        'actor_orientation': []
    }
}

Current Code:
```python
class gpt_{dual_bottles_pick_hard}({dual_bottles_pick_hard}):
    def play_once(self):
        pass
```
~~~

D.2. Introduction of Available APIs

```
Available API:
    "open_left_gripper": Open the left gripper to a specified position.,
    "close_left_gripper": Close the left gripper to a specified position.,
    "open_right_gripper": Open the right gripper to a specified position.,
    "close_right_gripper": Close the right gripper to a specified position.,
    "together_open_gripper": Open both left and right grippers to specified positions.,
    "together_close_gripper": Close both left and right grippers to specified positions.,

    "left_move_to_pose_with_screw":
        def left_move_to_pose_with_screw(pose).
        Plan and execute a motion for the left arm using screw motion interpolation.
        No Return.
        Args:
        pose: list [x, y, z, qw, qx, qy, qz], the target pose of left end-effector,
    "right_move_to_pose_with_screw":
        def right_move_to_pose_with_screw(pose).
        Plan and execute a motion for the right arm using screw motion interpolation.
        No Return.
        Args:
        pose: list [x, y, z, qw, qx, qy, qz], the target pose of right end-effector,
    "together_move_to_pose_with_screw":
        def together_move_to_pose_with_screw(left_target_pose, right_target_pose).
        Plan and execute motions for both left and right arms using screw motion interpolation.
        No Return.
        Args:
        left_target_pose: list [x, y, z, qw, qx, qy, qz], the target pose of left end-effector
        right_target_pose: list [x, y, z, qw, qx, qy, qz], the target pose of right end-effector,

    "get_actor_functional_pose":
        def get_actor_functional_pose(actor, actor_data),
        Get the functional pose of the actor in the world coordinate system.
        Returns: pose: list [x, y, z, qw, qx, qy, qz].
        Args:
        actor: Object(self.actor), the object of actor in render.
        actor_data: dict(self.actor_data), the actor_data match with actor.,

    "get_grasp_pose_to_grasp_object":
        def get_grasp_pose_to_grasp_object(self, endpose_tag: str, actor, actor_data = DEFAULT_ACTOR_DATA,
     pre_dis = 0),
        This function is used to grasp actor from the labeled contact points of the actor, and return the
     most suitable pose of the end-effector.
        Returns: pose: list [x, y, z, qw, qx, qy, qz].
        Args:
            endpose_tag: str, the endpose tag of the actor, can be 'left' or 'right'.
            actor: Object(self.actor), the object of actor in render.
            actor_data: dict(self.actor_data), the actor_data match with actor.
            pre_dis: float, the distance between grasp pose and target actor pose.,

    "get_grasp_pose_from_goal_point_and_direction":
        def get_grasp_pose_from_goal_point_and_direction(self, actor, actor_data,  endpose_tag: str,
     actor_functional_point_id, target_point, target_approach_direction, actor_target_orientation = None,
     pre_dis):
        This function is used to move the actor's point of action to the target point when the direction of
     the end-effector is given, return the pose of the end-effector.
        The actor refers to an object being grasped by robotic grippers. actor_target_orientation is the
     orientation of the actor after grasping.
        Returns: pose: list [x, y, z, qw, qx, qy, qz].
        Args:
        actor: Object(self.actor), the object of actor in render.
        actor_data: dict(self.actor_data), the actor_data match with actor.
        endpose_tag: str, the endpose tag of the actor, can be 'left' or 'right'.
        actor_functional_point_id: int, the index of the functional point of the actor.
        target_point: list [x, y, z], the target point pose which the actor's target_pose expected to move to.
        target_approach_direction: list [qw, qx, qy, qz], the approach direction which the actor's expected
     approach direction at the target point.
        The target approach direction can use self.world_direction_dic['left', 'front_left', 'front',
     'fron_right', 'right', 'top_down'].
        actor_target_orientation: list [x, y, z], the orientation of the actor after grasping. The positive
     directions of the xyz axis are right, front, and up respectively. You can give a direction vector to
     specify the target direction of the object. like [0, 0, 1] means the actor' orientation is up and [0, 1,
     0] means the actor's orientation is front.
        pre_dis: float, the distance on approach direction between actor's point of action and target point.,

    "get_avoid_collision_pose":
        def get_avoid_collision_pose(self, avoid_collision_arm_tag: str),
        This function can obtain the safe position of the specified robot arm to avoid collision when both
     arms need to move at the same time.
        Returns: pose: list [x, y, z, qw, qx, qy, qz].
        Args:
        avoid_collision_arm_tag: str, 'left' or 'right'.,

    "get_actor_goal_pose":
        def get_actor_goal_pose(self, actor, actor_data, id),
        This function is used to get the target pose point of an actor in world axis.
        Returns: pose: list [x, y, z].
        Args:
        actor: Object(self.actor), the object of actor in render.
        actor_data: dict(self.actor_data), the actor_data match with actor.
        id: int, the id of the actor, if the actor has multiple target points. And default is 0.,
```

D.3. Function Example

~~~
Function Example:
    You can retrieve the actor object by the actor's name:
    ```python
    actor = self.actor_name_dic['actor_name']
    ~~~
    You can retrieve the actor_data object by the actor_data's name:
    ```python
    actor_data = self.actor_data_dic['actor_data_name']
    ```

    Here are some APIs and examples of grasping objects:
    If you want to get the gripper pose to grasp the actor, you typically execute the following code:
    ```python
    grasp_pose = self.get_grasp_pose_to_grasp_object(endpose_tag = "left", self.actor, self.actor_data,
     pre_dis = 0.09)  # endpose_tag can be "left" or "right"
    ```

    If you want to pick up an actor, you can refer to the following sample code:
    ```python
    pre_grasp_pose = self.get_grasp_pose_to_grasp_object(endpose_tag = "left", self.actor, self.actor_data,
     pre_dis = 0.09)  # endpose_tag can be "left" or "right"
    target_grasp_pose = self.get_grasp_pose_to_grasp_object(endpose_tag = "left", self.actor,
     self.actor_data, pre_dis = 0)  # endpose_tag can be "left" or "right"
    self.left_move_to_pose_with_screw(pre_grasp_pose)      # left arm move to the pre grasp pose
    self.left_move_to_pose_with_screw(target_grasp_pose)   # left arm move to the grasp pose
    self.close_left_gripper()                              # close left gripper to grasp the actor
    self.left_move_to_pose_with_screw(pre_grasp_pose)      # lift the actor up
    ```
    The code for grasping with the right arm or both arms is similar to the above code.

    For the grasping of a certain actor, the movement of the end-effector typically executes the following
     codes:
    ```python
    actor_pose = self.get_actor_goal_pose(self.actor, self.actor_data)

    if actor_pose[0] > 0:           # if the actor in the right side, use right arm to grasp the actor
        # grasp actor with right arm
    else:                           # if the actor in the left side, use left arm to grasp the actor
        # grasp actor with left arm
    ```

    Here are some examples of gripper control:
    ```python
    self.close_left_gripper(pos = 0.02)     # Close half of the left gripper
    self.close_left_gripper(pos = -0.01)    # Tighten the left gripper.
    self.open_left_gripper(pos = 0.02)      # Open half of the left gripper
    self.close_right_gripper(pos = 0.02)    # Close half of the right gripper
    self.close_right_gripper(pos = -0.01)   # Tighten the right gripper.
    self.open_right_gripper(pos = 0.02)     # Open half of the right gripper
    self.together_close_gripper(left_pos = 0.02,right_pose = 0.02) # Together close half of grippers
    ```
    Note:
    For grabbing some objects, you may need to close the clamping jaws tightly to grab them. You can adjust
     this through the 'pos' parameter, like 'pos = -0.01'.
    By default 'pos' is 0, when close gripper.
```
```

Here are some APIs and examples of moving objects:
Note: The drop height of the actor depends on the distance of the actor that was lifted up the previous action.
To move an object to the target point, the 'get_grasp_pose_from_goal_point_and_direction()' is often called first to obtain the target's gripper posture.

If you want to move the point of actor which is grasped by the gripper action to the target point, you typically execute the following code:
```python
pre_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_pose, target_approach_direction, pre_dis =
 0.09)
target_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_pose, target_approach_direction, pre_dis = 0)
self.left_move_to_pose_with_screw(pre_grasp_pose)      # left arm move to the pre grasp pose
self.left_move_to_pose_with_screw(target_grasp_pose)   # left arm move to the grasp pose
self.open_left_gripper()  # open left gripper to place the target object
# You also can move right arm
```
Note:
1. The target_approach_direction is the approach direction which the actor's expected approach direction at the target point.
2. actor_functional_point_id is the index of the functional point of the actor, You can choose based on the given function points information.
3. For the parameter target_approach_direction, you can use self.world_direction_dic['left', 'front_left', 'front', 'fron_right', 'right', 'top_down'].
4. The target pose can be obtained by calling the 'get_actor_goal_pose()' function.

If you also have requirements for the target orientation of the object, you can specify the actor_target_orientation parameter through the direction vector to determine the final orientation of the object:
```python
# the actor target orientation is front, the direction vector is [0,1,0]
# The positive directions of the direction vector xyz axis are right, front, and up respectively.
pre_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_pose, actor_target_orientation = [0,1,0],
 target_approach_direction, pre_dis = 0.09)
target_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_pose, actor_target_orientation = [0,1,0],
 target_approach_direction, pre_dis = 0)
self.left_move_to_pose_with_screw(pre_grasp_pose)      # left arm move to the pre grasp pose
self.left_move_to_pose_with_screw(target_grasp_pose)   # left arm move to the grasp pose
self.open_left_gripper()  # open left gripper to place the target object
```

If you need to align the functional axis of the grabbed object with the functional axis of the target object, you can use the following code:
```python
target_actor_functional_pose = self.get_actor_functional_pose(self.actor, self.actor_data,
 actor_functional_point_id = 0)
target_actor_point = target_actor_functional_pose[:3]
target_approach_direction = target_actor_functional_pose[3:]
pre_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_point = target_actor_point,
 target_approach_direction = target_approach_direction, pre_dis = 0.09)
target_grasp_pose = self.get_grasp_pose_from_goal_point_and_direction(self.actor, self.actor_data,
 endpose_tag = "left", actor_functional_point_id = 0, target_point = target_actor_point,
 target_approach_direction = target_approach_direction, pre_dis = 0)
self.left_move_to_pose_with_screw(pre_grasp_pose)      # left arm move to the pre grasp pose
self.left_move_to_pose_with_screw(target_grasp_pose)   # left arm move to the grasp pose
self.open_left_gripper()  # open left gripper to place the target object
```
Note:
1. The parameter actor in get_grasp_pose_from_goal_point_and_direction() should be grasp actor, not the target actor.
2. self.world_direction_dic is a dict of different approach directions.
3. This situation usually occurs when hanging objects or performing some delicate operations.
4. actor_functional_point_id is the index of the functional point of the actor, You can choose based on the given function points information.

Some tasks involve simultaneous operations of the left and right arms, which may require calling the collision avoidance function:
1. There is no need to avoid collision at the end of the task.
2. If both arms have moved at the same time before, and the next step needs to be to move the left arm first to place the target object, You can first obtain the pose of the right arm that can avoid subsequent collisions, and then move both arms at the same time:

~~~
```python
# Get left and right arm target pose
# Here, the direction in which the object contacts the target point is vertically top_down as an example.
# The actor target orientation is left, the direction vector is [-1,0,0].
left_pre_pose = self.get_grasp_pose_from_goal_point_and_direction(left_actor, left_actor_data,
 endpose_tag="left", actor_functional_point_id = 0, target_point=point1,
 target_approach_direction=self.world_direction_dic['top_down'], actor_target_orientation=[-1, 0, 0],
 pre_dis=0.05)
left_target_pose = self.get_grasp_pose_from_goal_point_and_direction(left_actor, left_actor_data,
 endpose_tag="left", actor_functional_point_id = 0, target_point=point1,
 target_approach_direction=self.world_direction_dic['top_down'], actor_target_orientation=[-1, 0, 0],
 pre_dis=0)
right_pre_pose = self.get_grasp_pose_from_goal_point_and_direction(right_actor, right_actor_data,
 endpose_tag="right", actor_functional_point_id = 0, target_point=point2,
 target_approach_direction=self.world_direction_dic['top_down'], actor_target_orientation=[-1, 0, 0],
 pre_dis=0.05)
right_target_pose = self.get_grasp_pose_from_goal_point_and_direction(right_actor, right_actor_data,
 endpose_tag="right", actor_functional_point_id = 0, target_point=point2,
 target_approach_direction=self.world_direction_dic['top_down'], actor_target_orientation=[-1, 0, 0],
 pre_dis=0)
# right arm avoid collision pose
right_avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag = 'right')
# move left arm to the pre pose and right arm to the avoid collision pose
self.together_move_to_pose_with_screw(left_pre_pose, right_avoid_collision_pose)
# put down the actor on left gripper
self.left_move_to_pose_with_screw(left_target_pose)
self.open_left_gripper()  # open left gripper to place the target object
# left arm avoid collision pose
left_avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag = 'left')
# move right arm to the target pose and left arm to the avoid collision pose
self.together_move_to_pose_with_screw(left_avoid_collision_pose, right_pre_pose)
# put down the actor on right gripper
self.right_move_to_pose_with_screw(right_target_pose)
self.open_right_gripper()  # open right gripper to place the target object
# avoid_collision_arm_tag: 'left' or 'right'
# direction: 'left', 'right', 'front', 'back', 'up', 'down'
```
Note:
1. If the move_arm_tag is 'left', the direction also not be 'right', and same for 'right'.
2. Collision avoidance may only be necessary if both arms have been moved.
~~~

D.4. Sample Code for Blocks Stack Hard Task

```python
class gpt_blocks_stack_hard(blocks_stack_hard):
    def play_once(self):
        # Retrieve actor objects
        block1 = self.actor_name_dic['block1']
        block2 = self.actor_name_dic['block2']
        block3 = self.actor_name_dic['block3']
        block1_target_pose = self.actor_name_dic['block1_target_pose']

        # Retrieve actor data objects
        block1_data = self.actor_data_dic['block1_data']
        block2_data = self.actor_data_dic['block2_data']
        block3_data = self.actor_data_dic['block3_data']
        block1_target_pose_data = self.actor_data_dic['block1_target_pose']

        # Define pre-dis for grasping and placing
        pre_dis = 0.08

        # Function to grasp and place a block
        def grasp_and_place(block, block_data, target_pose, target_pose_data, pre_dis):
            # Determine which arm to use based on the block's x coordinate
            block_pose = self.get_actor_goal_pose(block, block_data)
            if block_pose[0] > 0:
                arm_tag = "right"
                move_function = self.right_move_to_pose_with_screw
                close_gripper_function = self.close_right_gripper
                open_gripper_function = self.open_right_gripper
            else:
                arm_tag = "left"
                move_function = self.left_move_to_pose_with_screw
```

```
                close_gripper_function = self.close_left_gripper
                open_gripper_function = self.open_left_gripper

            # Get the grasp pose
            pre_grasp_pose = self.get_grasp_pose_to_grasp_object(endpose_tag=arm_tag, actor=block,
actor_data=block_data, pre_dis=pre_dis)
            target_grasp_pose = self.get_grasp_pose_to_grasp_object(endpose_tag=arm_tag, actor=block,
actor_data=block_data, pre_dis=0)

            # Move to the pre-grasp pose
            move_function(pre_grasp_pose)

            # Move to the grasp pose
            move_function(target_grasp_pose)

            # Close the gripper to grasp the block
            close_gripper_function()

            # Lift the block up
            move_function(pre_grasp_pose)

            # Get the target pose for placing the block
            target_point = self.get_actor_goal_pose(target_pose, target_pose_data)
            target_approach_direction = self.world_direction_dic['top_down']
            pre_place_pose = self.get_grasp_pose_from_goal_point_and_direction(block, block_data,
endpose_tag=arm_tag, actor_functional_point_id=0, target_point=target_point,
target_approach_direction=target_approach_direction, pre_dis=pre_dis)
            target_place_pose = self.get_grasp_pose_from_goal_point_and_direction(block, block_data,
endpose_tag=arm_tag, actor_functional_point_id=0, target_point=target_point,
target_approach_direction=target_approach_direction, pre_dis=0)

            # Move to the pre-place pose
            move_function(pre_place_pose)

            # Move to the place pose
            move_function(target_place_pose)

            # Open the gripper to place the block
            open_gripper_function()

            # Lift the arm up
            move_function(pre_place_pose)

        # Grasp and place block1
        grasp_and_place(block1, block1_data, block1_target_pose, block1_target_pose_data, pre_dis)

        # Avoid collision if necessary
        if self.get_actor_goal_pose(block1, block1_data)[0] > 0:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='left')
            self.left_move_to_pose_with_screw(avoid_collision_pose)
        else:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='right')
            self.right_move_to_pose_with_screw(avoid_collision_pose)

        # Grasp and place block2 on top of block1
        grasp_and_place(block2, block2_data, block1, block1_data, pre_dis)

        # Avoid collision if necessary
        if self.get_actor_goal_pose(block2, block2_data)[0] > 0:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='left')
            self.left_move_to_pose_with_screw(avoid_collision_pose)
        else:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='right')
            self.right_move_to_pose_with_screw(avoid_collision_pose)

        # Grasp and place block3 on top of block2
        grasp_and_place(block3, block3_data, block2, block2_data, pre_dis)

        # Avoid collision if necessary
        if self.get_actor_goal_pose(block3, block3_data)[0] > 0:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='left')
            self.left_move_to_pose_with_screw(avoid_collision_pose)
        else:
            avoid_collision_pose = self.get_avoid_collision_pose(avoid_collision_arm_tag='right')
            self.right_move_to_pose_with_screw(avoid_collision_pose)
```