RoboFactory: Exploring Embodied Agent Collaboration with Compositional Constraints

Yiran Qin[1,2*],   Li Kang[2,6*],   Xiufeng Song[2,7*],   Zhenfei Yin[3†],
Xiaohong Liu[7],   Xihui Liu[4],   Ruimao Zhang[5†],   Lei Bai[2†]

[1]The Chinese University of Hong Kong, Shenzhen   [2]Shanghai Artificial Intelligence Laboratory
[3]Oxford   [4]HKU   [5]Sun Yat-sen University   [6]Tongji University   [7]Shanghai Jiao Tong University

yiranqin@link.cuhk.edu.cn   faceong02@gmail.com   akikaze@sjtu.edu.cn
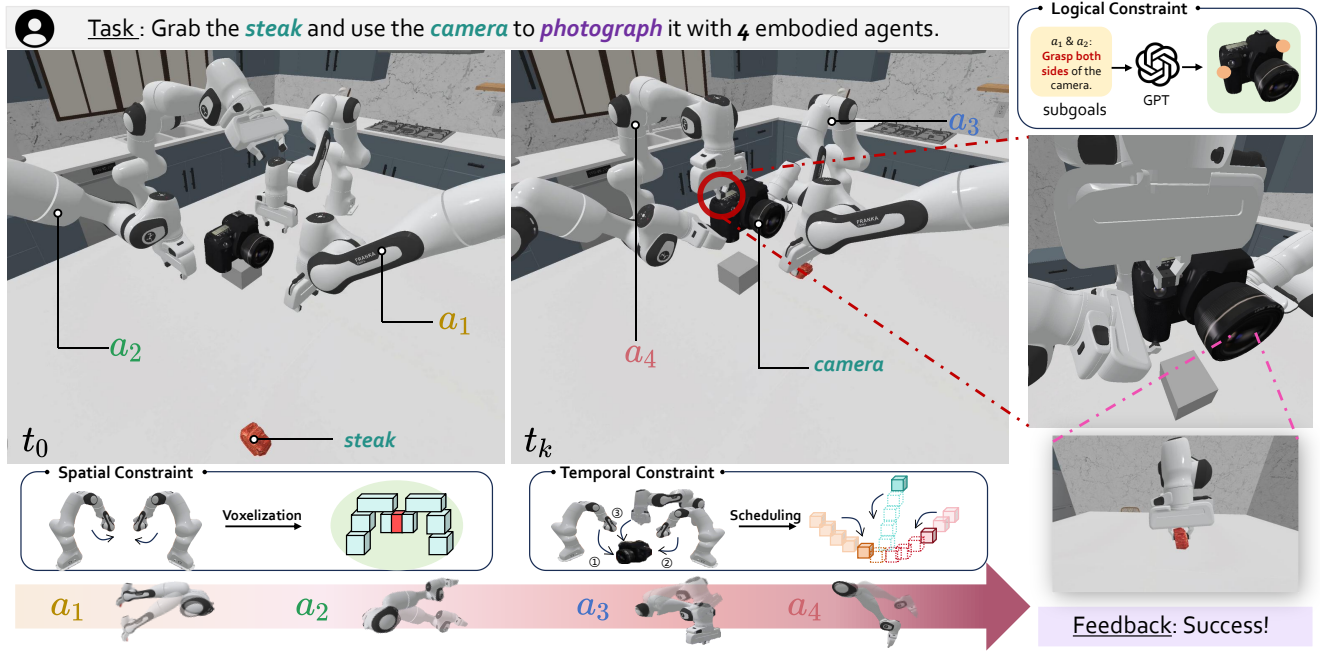
https://iranqin.github.io/robofactory/

Figure 1. When performing the task "Grab the steak and use the camera to photograph it with 4 embodied agents", collaboration among multiple agents is required: a_1 grasps the steak, a_2 and a_3 lift the camera, and a_4 presses the shutter to take the photo. However, each agent cannot focus solely on its own task. We introduce the concept of compositional constraints to ensure safe and efficient collaboration among the agents. Logical constraints prevent incorrect interaction forms (e.g., a_3 grabbing the camera lens, causing damage). Spatial constraints avoid catastrophic hardware damage (e.g., collisions between a_2 and a_3 during trajectory execution). Temporal constraints prevent inefficient collaboration (e.g., a_1 waiting unnecessarily due to nonexistent collisions while other agents execute their tasks).

Abstract

Designing effective embodied multi-agent systems is critical for solving complex real-world tasks across domains. Due to the complexity of multi-agent embodied systems, existing methods fail to automatically generate safe and efficient training data for such systems. To this end, we pro-pose the concept of compositional constraints for embodied multi-agent systems, addressing the challenges arising from collaboration among embodied agents. We design various interfaces tailored to different types of constraints, enabling seamless interaction with the physical world. Leveraging compositional constraints and specifically designed interfaces, we develop an automated data collection framework for embodied multi-agent systems and introduce the first benchmark for embodied multi-agent manipulation, Robo-

---

* Equal contribution   † Corresponding author

Factory. Based on RoboFactory benchmark, we adapt and evaluate the method of imitation learning and analyzed its performance in different difficulty agent tasks. Furthermore, we explore the architectures and training strategies for multi-agent imitation learning, aiming to build safe and efficient embodied multi-agent systems. Please see the project page at https://iranqin.github.io/robofactory/.

1. Introduction

With the increasing diversity of robotic forms and the advancement of robotic control strategies, current robotic systems are capable of performing fixed tasks [6, 49] or executing tasks based on instructions [2, 19], offering infinite possibilities to build interactive agents in the real world. While these robotic systems typically focus on single-agent tasks, many real-world applications—such as manufacturing and medical assistance—require multiple embodied agents to collaborate on tasks. Such collaboration is vital for tackling tasks that exceed the capabilities of a single agent and enhancing task efficiency through multi-agent deployment.

To train the policies of multi-embodied agents, researchers often need to collect data through simultaneous remote operation by multiple people, which is extremely inefficient. With the rise of large language models (LLMs) [1, 36], many single-agent works have attempted to leverage their powerful reasoning capabilities to automate the data generation process [28, 29], using predefined motion primitives to carry out physical interactions with the environment. This significantly reduces the labor and time costs associated with data collection. However, when it comes to the more challenging tasks of multi-agent collaboration, the dimensions that need to be considered become far more complex: 1) Plan global tasks and allocate agents based on task logic and agent availability, ensuring logical consistency and maximizing efficiency. 2) Manage shared physical space to prevent collisions between embodied agents. 3) Schedule agents sharing the same space at different timesteps or perform some low-level operations simultaneously to improve system efficiency. Simple adaptations of single-agent embodied systems cannot meet such requirements, and additional constraints need to be introduced.

We first introduce *compositional constraints* based on three specific constraint for multi-agent embodied data generation. Each constraint is specifically designed for multi-agent collaborative tasks and restricts the behavior of embodied agents in the corresponding dimension (e.g., space). By combining these constraints, agents could conduct more reasonable and safe behaviors that aligns with real-world embodied agent collaboration scenarios: 1) **Logical Constraints**: Define the rules and relationships that govern

valid robot behaviors. 2) **Spatial Constraints**: Specify constraints based on physical or spatial boundaries. 3) **Temporal Constraints**: Impose constraints related to time to make more efficient collaboration.

In this work, we propose a framework *RoboFactory* for multi-agent embodied data generation based on *compositional constraints*, which not only benefits from the generalization power of LLMs but also satisfies the additional requirements of multi-agent tasks. Given the global task description, prior information, and observations, RoboBrain generates the next sub-goals for each agent, outputs textual compositional constraints, and produces unconstrained trajectory sequences for each agent by invoking predefined motion primitives to achieve these sub-goals. However, the compositional constraints generated by RoboBrain are limited to the textual modality, making them insufficient for directly constraining real-world decision-making. To address this limitation, we designed various interfaces tailored to different types of constraints, enabling seamless interaction with the physical world. As a result, RoboChecker can construct corresponding constraint interfaces based on these textual constraints and the current multi-agent state to ensure agents do not violate any constraints while executing the generated trajectories.

Within this framework, *RoboFactory* addresses the challenges of scaling up from single-agent embodied systems to multi-agent embodied systems by constructing a safe and efficient data production pipeline. Using RoboFactory, we designed various multi-agent collaboration scenarios and evaluated the design of embodied multi-agent systems within these settings. First, we validated the necessity of compositional constraints in multi-agent data generation. We then deployed the diffusion policy on several multi-agent collaborative tasks and conducted extensive testing. Furthermore, we explored the design of multi-agent systems based on imitation learning, investigating more effective architectural designs. Our contributions can be summarized as follows:

1. We propose the concept of compositional constraints for embodied multi-agent systems, addressing the challenges arising from collaboration among embodied agents.
2. Leveraging compositional constraints and specifically designed interfaces, we develop an automated data collection framework for embodied multi-agent systems and introduce the first benchmark for embodied multi-agent manipulation, *RoboFactory*.
3. Based on *RoboFactory*, we deploy imitation learning methods and conduct evaluations, and explore the architectures and training strategies for multi-agent imitation learning, aiming to build safe and efficient embodied multi-agent systems.

2. Related Work

2.1. Multi-Agent System

Multi-agent systems consist of multiple autonomous entities, each with access to distinct information and potentially conflicting objectives. Based on their functionalities, recent multi-agent systems can generally be categorized into two types: tool-based agent assistants [12, 21, 27, 40, 41, 45, 48] and simulation environments for societies or games [13, 33, 46, 52]. Different from the above work, *RoboFactory* focuses more on the application of multi-agent collaboration in real-world decision-making, especially the low-level manipulation of embodied agents.

2.2. Robot Manipulation

Behavioral Cloning (BC) [8, 16, 17, 24, 26] trains policies using pre-recorded human demonstrations to directly imitate expert behaviors, whereas Offline Reinforcement Learning (ORL) [4, 18, 20] refines action selection through reward maximization across extensive datasets. Generative approaches have expanded methodology: Action Chunking with Transformers (ACT) employs Transformer architectures combined with conditional variational autoencoders to model sequential decision-making [3, 37, 49]. Meanwhile, diffusion-based frameworks have gained traction in robotic imitation learning for their superior generation performance. Notable examples include Diffusion Policy [6] and its 3D variant [47] that utilizes point cloud observations to improve geometric understanding. Demonstration acquisition primarily relies on human-operated robotic systems across diverse tasks [9, 16, 25, 26], with simulator-based trajectory synthesis providing supplementary data sources [15, 29–32, 35]. Although these robotic systems demonstrate the ability to generate data, they have rarely explored effectively multi-agent robotic manipulation.

2.3. Visual Programming

Executing visual programming necessitates robust comprehension of visual concepts and spatio-temporal reasoning capabilities. While existing approaches demonstrate broad applicability in zero-shot scenario [7, 10, 22, 34, 38, 50] through fusion of large language models (LLMs) with vision systems, they often sacrifice granular precision in task execution. Contemporary methods [14, 28, 39] investigate vision-language models (VLMs) for visual programming. CaM [51] introduces constraint-based elements for program synthesis. In our work, *RoboFactory* generates robot trajectories through action primitives, further regulating the effectiveness of visual programming via compositional constraint interface.

3. Compositional Constraints

Constraints define practical and efficient boundaries grounded in real-world conditions. In multi-robot collaboration scenarios, the more complex decision space requires various constraints to ensure safety and effectiveness. Below, we propose three categories of common constraints that are crucial for modeling real-world boundaries in multi-embodied agent decision-making.

Logical Constraints. Logical Constraints define the permissible actions and interaction rules for agents, focusing on high-level logic such as interaction objects, contact points, and movement directions rather than the timing or sequence of operations. These constraints encode structural rules within task scenarios, such as usage permissions for interactive objects (e.g., only specific tools can be used for processing certain materials), contact point restrictions (e.g., agents must grasp objects from designated points), and directional consistency (e.g., when multiple agents transport an object, their applied forces must remain aligned). By formalizing interaction relationships—such as action compatibility, interaction constraints, and spatial coordination requirements—logical constraints ensure agents follow coherent operational procedures.

Spatial Constraints. Spatial Constraints defines where agents can operate and how physical interactions are structured. These include geometric boundaries (e.g., no agent may enter a 1-meter radius around active machinery), collaborative workspace partitioning (e.g., dividing a construction site into exclusive zones to prevent collisions), and task-specific placement requirements (e.g., components must be positioned within 2 cm of their target coordinates for valid assembly). They also govern adaptive spatial behaviors, such as dynamic rerouting around newly detected obstacles or adjusting gripper orientations to fit narrow apertures. By isolating physical feasibility from temporal and logical considerations, these constraints ensure agents operate within safe, structurally coherent environments.

Temporal Constraints. Temporal Constraints regulate when and in what order actions must be executed, addressing synchronization, deadlines, and sequential dependencies. These constraints ensure that agents align their behaviors with time-sensitive requirements, such as enforcing phased workflows (e.g., Agent C must wait 5 seconds after Agent D finishes welding to begin inspection) or coordinating parallel actions with strict time windows (e.g., two agents must lift an object simultaneously within a 0.5-second tolerance). They also manage dynamic adjustments, such as extending task durations in response to environmental delays or rescheduling actions when prior steps overrun. Unlike logical constraints, which define decision va-
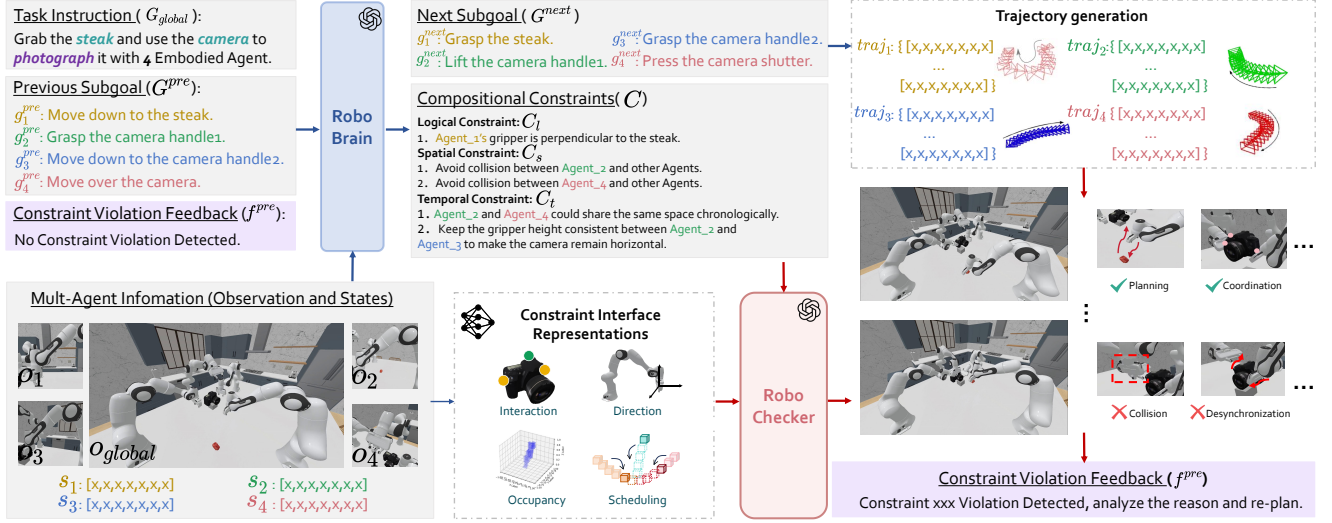
Figure 2. Overview of *RoboFactory*. Given the global task description, prior information, and observations, RoboBrain generates the next sub-goals for each agent and outputs textual compositional constraints. It then generates unconstrained trajectory sequences for each agent to achieve the corresponding sub-goals, invoking predefined motion primitives. RoboChecker constructs corresponding constraint interfaces based on the textual compositional constraints and the current multi-agent state. It checks whether the agents violate any constraints while executing the generated trajectories. This framework ensures the generation of safe and efficient collaborative data for multi-embodied agents by transforming abstract textual constraints into representations that can interact with agent behaviors through the construction of constraint interfaces.

lidity, temporal constraints focus strictly on timing feasibility—ensuring actions occur neither prematurely nor too late to maintain safety and efficiency.

Compositional Constraints. The collaboration of multi-embodied agents relies on the integration of logical, temporal, and spatial constraints. Logical constraints define interaction protocols and shared objectives, temporal constraints synchronize actions with task dependencies, and spatial constraints encode geometric and semantic boundaries. Together, these constraints balance decentralized autonomy with global coherence, enabling conflict resolution, resource optimization, and adaptability. This unified framework ensures that local decisions converge into robust, efficient, and executable collaborative behaviors.

4. RoboFactory

We first give an overview of the proposed *RoboFactory* (Sec. 4.1). Then, we elaborate on the Constraint Interface Generation (Sec. 4.2). Finally, we present the Dataset and Benchmark of *RoboFactory* (Sec. 4.3).

4.1. Overview

The proposed RoboFactory framework consists of two core modules, RoboBrain and RoboChecker. Our work focuses on long-horizon multi-agent manipulation task instructions $\mathcal{G}_{\text{global}}$ (e.g., "Grab the steak and use the camera to pho-

tograph it with 4 Embodied Agents."), utilizing RGB observations $\mathcal{O} = \{o_{\text{global}}, o_1, ..., o_n\}$, which consist of one global view and multiple ego-centric views from n agents $\{a_1, a_2, ..a_n\}$ (e.g., $n = 4$ in this case). As illustrated in Figure 2, the RGB images $\mathcal{O}$, combined with the text instructions $\mathcal{G}_{\text{global}}$, the previous subgoal sets $\mathcal{G}^{\text{pre}} = \{g_1^{\text{pre}}, ..., g_n^{\text{pre}}\}$, and the Constraint Violation Feedback (e.g., subgoal collaboration success or constraint violation details) from RoboChecker, denoted as f^{pre}, are input into the RoboBrain $\mathcal{F}_{\text{VLM}}$ (e.g., GPT-4o [1]). The model then generates the next subgoal sets $\mathcal{G}^{\text{next}} = \{g_1^{\text{next}}, ..., g_n^{\text{next}}\}$ along with the corresponding textual compositional constraints $\mathcal{C} = \{\mathcal{C}_l, \mathcal{C}_s, \mathcal{C}_t\}$, each type of constraint set contains multiple specific constraints c (e.g., "Avoid collision between a_2 and other agents."). Here, $\mathcal{C}_l$ represents the logical constraint set that multi-agents must adhere to based on prior knowledge during abstract task decomposition and scheduling to complete sub-tasks (e.g., the action of taking a photo requires a specific agent to press the shutter button). $\mathcal{C}_s$ denotes the need for multi-agents to avoid collisions—both agent-object and agent-agent—when sharing common spaces during task collaboration. $\mathcal{C}_t$ reflects the temporal-spatial sharing strategy among agents. Specifically, agents occupying a shared space at different time steps can improve collaboration efficiency (e.g., a_2 occupies a space in the t_0, while a_4 can occupies the same space in the t_1, achieving temporal-spatial sharing). This process can be expressed as follows:
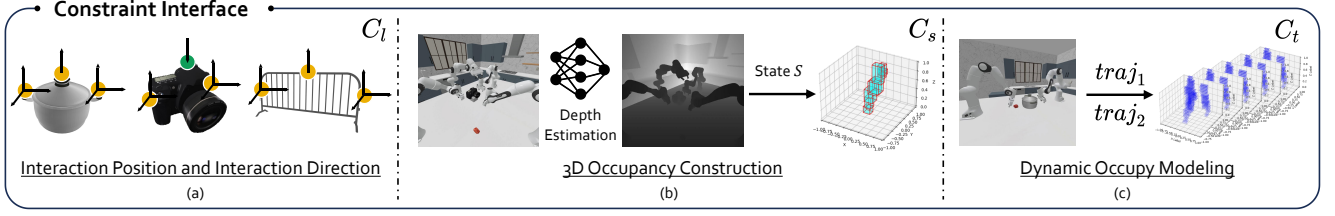
Figure 3. Different Constraint Interface. For C_l, we annotated the interactive points of objects and the interactive directions of each point. For C_s, we modeled observations to obtain depth maps and used them, along with the robotic arm states, to construct 3D occupancy representations. For C_t, we modeled temporal-state representations based on the trajectories of agents at each changing position and used these representations for scheduling through analysis.

$$\mathcal{G}^{\text{next}}, \mathcal{C} = \mathcal{F}_{\text{VLM}}(\mathcal{O}, \mathcal{G}_{\text{global}}, \mathcal{G}^{\text{pre}}, f^{\text{pre}}) \qquad (1)$$

RoboBrain RoboBrain then generates trajectory sequences $traj_1, ..., traj_n$ for each agent to complete the corresponding subgoals by leveraging visual programming to invoke predefined motion primitives; these trajectories are unconstrained and may have potential failure risks. The detailed trajectory generation process is provided in the supplementary materials. Each agent executes its trajectory sequence along the time dimension, operating serially within the agent and in parallel between agents. This process is continuously monitored by RoboChecker to ensure the logical, spatial, and temporal validity of the subgoal trajectories. The constraints generated by RoboBrain are confined to the textual space, making them ineffective for directly constraining trajectory data. Therefore, special interfaces are required to transform the textual constraints into specific representations that can directly interact with the real world and constrain the trajectory data.

RoboChecker In RoboChecker, we provide GPT-4o [1] with the textual constraints $\mathcal{C}$ for constraint-aware visual programming. For each constraint c_i (e.g., "Avoid collision between Agent_2 and other Agents"), a corresponding interface h_i which is define in Sec. 4.2 is created based on textual constraints c_i, addition with generated trajectory and other information needed (RGB images $\mathcal{O}$, robot states $\mathcal{S} = \{s_1, ...s_n\}$). Evaluation protocol is then generated (i.e., check code for trajectory), based on h_i and the textual constraints $\mathcal{C}$. This protocol evaluates whether the constraints are violated at the current time step by analyzing the interface h_i. It returns a boolean indicating whether a constraint violation has occurred and a string describing the reason for the violation. If the protocol returns False, trajectory execution halts immediately, and the accompanying string is used as feedback (f^{pre}) for re-planning. Otherwise, the subgoal is marked as completed. In either case, the process repeats. Fully validated trajectories and observation sequences are served as part of the RoboFactory benchmark dataset.

4.2. Constraint Interface

To enable compositional constraints to govern decision-making in real-world scenarios, we need to model the abstract textual constraints into concrete criteria that can interact with the real world called **Constraint Interface**. This ensures that RoboChecker can effectively restrict the set of generated trajectories. We have modeled the compositional constraints using four physical representation methods.

We model the logical constraint using two interaction logics in the physical world, namely interaction position and interaction direction, as shown in Fig. 3(a). **Interaction Position**: For each 3D asset, we annotate the interaction positions. Different positions represent different interaction logics. For instance, grasping a camera and using a camera have distinct interaction positions. **Interaction Direction**: Similarly, for every 3D asset, we mark the interaction directions. Different interaction behaviors between the robotic arm and the object follow different directional logics. For example, pressing the camera shutter requires the gripper of the robotic arm to move in the direct-facing direction of the shutter. In RoboChecker, based on the interaction form between the agent and the object, it will determine whether the current trajectory and grasping pose violate the logical constraints of interaction position and direction.

We analyze the current scene and establish a **3D occupancy** interface to implement spatial constraints, as shown in Fig. 3(b). Specifically, we conduct depth estimation of the current 3D scene by utilizing hardware devices (such as depth cameras) or depth estimation methods [43, 44]. Then, based on the current states of the robotic arm, we calculate the absolute coordinates of each joint point within the current space. By integrating the depth information, we obtain the occupancy information of the robotic arm and the objects. It is worth noting that we set a voxel with a size of 5cm*5cm*5cm as the basic discrete occupancy unit to reduce the computational cost. In RoboChecker, according to the occupancy relationship between the agent whose po-

Align Camera: Agent 1 **picks up** the *object*. Agent 2 and 3 **aligns the** *camera* to the object.

t_0 t_1 t_2 t_3

Place Food: Agent 1 **lifts** the *pot's lid*. Agent 2 **picks up** the *food* and **places** it in the *pot*.

t_0 t_1 t_2 t_3

Stack Cube: Agent 1 **places** the blue *cube* to the targe. Agent 2 **places** the red *cube* on it.
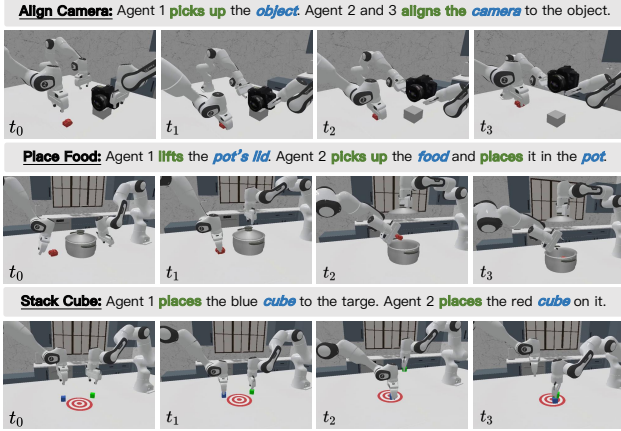
t_0 t_1 t_2 t_3

Figure 4. Demonstrations of the *RoboFactory* Benchmark.

sition is changing and other elements in the scene, it will determine whether a collision occurs and whether the spatial logic is violated.

We perform **dynamic occupancy modeling** for all intelligent agents that need to move under a sub-goal set $\mathcal{G}_{\text{next}}$ to account for temporal constraints, as shown in Fig. 3(c). In cooperative tasks involving multiple embodied intelligent agents, the behaviors of robotic arms often occur simultaneously. Consequently, relying solely on spatial occupancy constraints can result in irrational scheduling, which significantly reduces task completion efficiency. To address this, RoboChecker utilizes the temporal occupancy information of these intelligent agents to detect and prevent irrational scheduling as well as violations of temporal logic.

By establishing various interfaces required for compositional constraints, RoboChecker can convert abstract text-based constraints into representational forms that can interact with real-world decision-making. This enables the constraint of unrestricted trajectories, thereby generating safe and efficient multi-agent collaborative data.

4.3. Benchmark

Based on the methods described above, we propose the *RoboFactory* Benchmark, which is built on the ManiSkill simulator [35], an open-source platform for robot simulation. Tab. 1 demonstrates the comparison between other embodied benchmarks and the *RoboFactory* benchmark. Our *RoboFactory* benchmark features multi-agent tasks and the integration of high-level planning and low-level controlling. It includes 11 tasks across environments with varying numbers of agents, built based on the Franka Emika Panda Arm, a 7-DoF robotic manipulator equipped with an end-effector that enables flexible manipulation tasks, as depicted in Fig. 4. We utilize publicly available 3D assets from sources such as the PartNet-Mobility Dataset [42]. For each task scenario, we configured an ego-centric camera for each agent and a global camera for all agents.

Table 1. Comparisons between *RoboFactory* and other embodied benchmarks. It features multi-agent tasks and the integration of high-level planning and low-level controlling.

Benchmark	Single-agent	Multi-agent	Task Level
EgoPlan-Bench [5]	✓	✗	Plan
MMWorld [11]	✓	✗	Plan
VAB [23]	✓	✗	Plan
RoboCasa [31]	✓	✗	Plan
RoboTwin [29]	✓	✗	Plan & Control
RoboFactory(Ours)	✓	✓	Plan & Control

Our benchmark emphasizes efficient collaboration and coordination among agents in multi-agent environments. Agents must work together to complete specific tasks. For instance, in the task of Place Food, one agent must open the pot lid before another can place the food inside. The tasks involve diverse asset types, and the initial settings (e.g., asset placement) are randomly assigned using random seeds and can be easily replayed using the same seed. This domain randomization approach can effectively increase the diversity of training scenarios. In *RoboFactory* benchmark, 150 sets of data have been pre-collected for each task in form of camera RGB image observations, joint action of the robotic arms. More details can be found in the supplementary materials.

5. Experiment

Our experiments consist of following three parts: **(1)** the evaluation of Diffusion Policy on RoboFactory Benchmark (Sec. 5.1); **(2)** comparison of different architectural design of multi-agent systems based on imitation learning (Sec. 5.2); **(3)** ablation studies on different constraints in RoboFactory data generation (Sec. 5.3).

5.1. Evaluation of *RoboFactory* Benchmark

We evaluate Diffusion Policy (DP) [6], a generative method based on imitation learning, across 11 tasks in the *RoboFactory* benchmark to justify the effectiveness of our method. For each agent, an individual policy is trained by taking ego-centric observations in RGB as input, without considering the states and actions of other agents. We train the policies using 50, 100, and 150 expert demonstration data per task. More details for training strategies can be found in the supplementary materials.

Tab. 2 demonstrates the main performance on *RoboFactory* benchmark. First, the increase in success rates with additional training data emphasizes the necessity of RoboFactory in efficiently generating high-quality datasets. Specifically, tasks involving one, three, and four agents achieve optimal performance with 150 training demonstrations. These findings highlight the critical role of ample data, particularly in complex multi-agent environments. At the same time, tasks involving two agents achieve optimal perfor-

Table 2. DP baseline performance results. We report the success rate across benchmark tasks with different amounts of demonstration data.

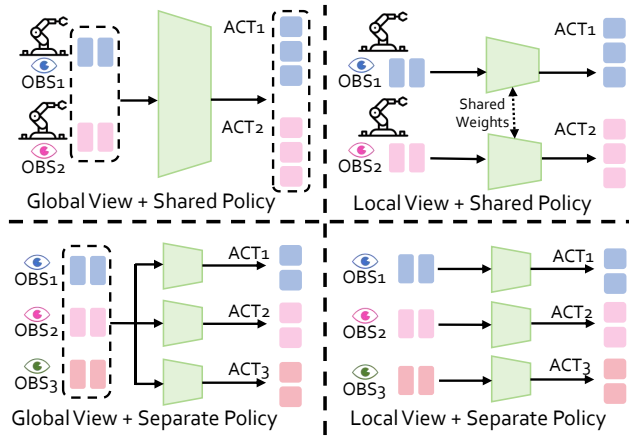| Task Level	Task Name	Success Rate		
		50 Demo	100 Demo	150 Demo
1-Agent	Pick Meat	32%	**61%**	58%
	Stack Cube	17%	38%	**44%**
	Strike Cube	26%	42%	**45%**
	Average	25%	47%	**49%**
2-Agent	Pass Shoe	9%	**20%**	12%
	Place Food	5%	**23%**	20%
	Lift Barrier	24%	**60%**	58%
	Two Robots Stack Cube	14%	**27%**	20%
	Average	13%	**32.5%**	27.5%
3-Agent	Camera Alignment	7%	10%	**19%**
	Three Robots Stack Cube	8%	2%	**22%**
	Average	7.5%	6%	**20.5%**
4-Agent	Take Photo	5%	8%	**20%**
	Long Pipeline Delivery	0%	0%	0%
	Average	2.5%	4%	**10%**

Figure 5. We design four multi-embodied agent imitation learning architectures. The **Global View** in the image input represents the observation containing all agents, and the **Local View** represents the ego-view observation of each agent. In policy training, **Shared Policy** indicates that all agents share a policy, and **Separate Policy** indicates that each agent trains an independent policy.

mance with only 100 demonstrations, which can be attributed to the relative simplicity of individual agent actions in these scenarios. Training with 150 demonstrations may have introduced overfitting, causing the model to learn unnecessary patterns that do not generalize well to testing environments. Besides, the decline in success rates with an increasing number of agents highlights the limitations of current methodologies. Specifically, tasks involving one, two, three, and four agents achieve average success rates of 49%, 27.5%, 20.5%, and 10%, respectively. This significant performance degradation in multi-agent tasks indicates

Table 3. Results of four multi-embodied agent imitation learning architectures. We report the success rate on two tasks.

Policy	View Scope	Lift Barrier	Place Food
Shared	Global	49%	5%
Shared	Local	4%	0%
Separate	Global	26%	17%
Separate	Local	**58%**	**20%**

the challenges in facilitating effective collaboration among multiple agents. Moreover, the 0% success rate in the task of Long Pipeline Delivery task highlights the shortcomings of diffusion policies in learning long-term temporal dependencies. These findings underscore significant opportunities for advancing imitation learning techniques in multi-agent systems to enhance performance.

5.2. Multi-agent Imitation Learning

In Sec. 5.1, we adapt the single-agent imitation learning framework to a multi-agent system, where each agent trains an independent policy based on its egocentric view. As illustrated in Figure 5, the architecture of multi-embodied agent imitation learning can be categorized into four types based on the observation space and policy sharing strategy:

Global View and Shared Policy (Arch1): All agents share the same global observation and use a single shared policy to produce a joint action sequence, which is then assigned to the corresponding agents.

Local View and Shared Policy (Arch2): Each agent has its own independent egoview observation, which is fed into the same shared policy (with shared parameters) to generate respective action sequences.

Global View and Separate Policy (Arch3): All agents

Table 4. Ablation study for different constraints. We report the success rate(%↑) of effective data generation.

Components			Task Name		
Logical	Spatial	Temporal	Lift Barrier	Three Robots Stack Cube	Take Photo
✓	✗	✗	80.2	62.5	37.1
✓	✗	✓	85.4	84.2	62.2
✓	✓	✗	95.2	92.7	53.8
✓	✓	✓	**97.5**	**98.9**	**88.2**

Table 5. Ablation study for different constraints. We report the average episode length(↓) of the generated effective data.

Components			Task Name		
Logical	Spatial	Temporal	Lift Barrier	Three Robots Stack Cube	Take Photo
✓	✗	✗	123	685	407
✓	✗	✓	92.8	452	238
✓	✓	✗	115	652	325
✓	✓	✓	**80.7**	**424**	**204**

share the same global observation, while each agent has its own separate policy (with unshared parameters) to generate individualized action sequences.

Local View and Separate Policy (Arch4): Each agent has its own independent egoview observation, and each agent uses its own separate policy (with unshared parameters) to generate individualized action sequences.

We employ Arch4 as the pipeline to test all tasks in the RoboFactory benchmark. Additionally, we select two representative two-agent collaborative tasks—Lift Barrier and Food Place—to compare and analyze the four architectures. The experimental results are summarized in Table 1. By comparing the first and second rows of Table 1, we observe that when a single shared policy needs to learn strategies for multiple agents from different ego-views, it must infer the agent ID currently executing an action and generate the corresponding action. This challenging setup causes the shared policy to struggle and leads to degraded performance (49%-5%, 5%-0%). By comparing the first two rows (shared policy) with the third and fourth rows (separate policy), we find that Separate Policy achieves better performance in the Food Place task (5%-17%, 0%-20%). This improvement may be attributed to the Food Place task requiring distinct skills for the two robotic arms, where separate policies can specialize in learning the respective skills, thereby enhancing collaborative performance. Finally, by comparing the third and fourth rows, we observe that using Local View under the Separate Policy setup improves task success rates (26%-58%, 17%-20%). We hypothesize that this is because the egoview provides richer and more detailed information, enabling the policy to better handle fine-grained manipulations.

In summary, we design multiple multi-agent imitation learning architectures and conducted extensive experiments and analyses. We hope these findings can provide valuable insights for the future design of multi-agent imitation learning frameworks or multi-agent vision-language-action models, ultimately advancing the field of multi-embodied agent manipulation systems.

5.3. Ablation Study

Our ablation study aims to address two key questions: (1) Can the proposed constraints, particularly the compositional constraints, effectively improve the success rate of data generation (where higher success rates indicate faster data production)? (2) Do the proposed constraints lead to higher-quality data? We used the average episode length of the data as a metric to evaluate the quality. For tasks that are successfully executed, shorter data lengths suggest that agents can cooperate more effectively. In addition, shorter data lengths contribute to faster training and inference times. Tab. 4 and Tab. 5 demonstrate the ablation studies on benchmark.

Spatial and temporal constraints significantly improve task success rates. The absence of spatial constraints leads to frequent robotic arm collisions during task execution, drastically reducing the success rate. Additionally, without spatial constraints, the robotic arms lack corrective spatial feedback, failing to adjust their positions properly. For temporal constraints, the decline in success rate mainly stems from two factors. First, tasks requiring simultaneous execution (e.g., Lift Barrier) fail due to improper synchronization. Second, errors in the execution order of robotic actions occur, such as incorrect stacking sequences in the Three Robot Stack Cube task.

Temporal constraints play an important role in enhancing data quality. Temporal constraints streamline task execution by identifying opportunities for simultaneous robotic arm operations, facilitating parallel execution, and reducing episode length in the dataset. Moreover, by analyzing and scheduling the sequence of robotic actions, they contribute to generating more structured and efficient data. When incorporated with occupancy grids, temporal constraints make fine-grained spatial awareness at each timestep, reducing erroneous failure feedback and fostering greater data diversity.

6. Conclusion

We propose compositional constraints to tackle scalability challenges in transitioning from single-agent to multi-agent

embodied systems and leverage compositional constraints to develop an automated data collection framework *Robo-Factory*. We introduce the first benchmark for embodied multi-agent manipulation. By deploying imitation learning and evaluating policy architectures on this benchmark, we systematically explore training strategies to advance safe and efficient multi-agent systems.

Limitation While *RoboFactory* shows notable effectiveness, the constraints may struggle to accurately model intricate physical phenomena, potentially limiting their applicability in tasks requiring precise interactions.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023. 2, 4, 5

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi0: A vision-language-action flow model for general robot control. arXiv preprint arXiv:2410.24164, 2024. 2

[3] Thanpimon Buamanee, Masato Kobayashi, Yuki Uranishi, and Haruo Takemura. Bi-act: Bilateral control-based imitation learning via action chunking with transformer. In 2024 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), pages 410–415. IEEE, 2024. 3

[4] 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

[5] Yi Chen, Yuying Ge, Yixiao Ge, Mingyu Ding, Bohao Li, Rui Wang, Ruifeng Xu, Ying Shan, and Xihui Liu. Egoplan-bench: Benchmarking egocentric embodied planning with multimodal large language models. CoRR, 2023. 6

[6] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. The International Journal of Robotics Research, page 02783649241273668, 2023. 2, 3, 6

[7] Jaemin Cho, Abhay Zala, and Mohit Bansal. Visual programming for text-to-image generation and evaluation. In NeurIPS, 2023. 3

[8] Murtaza Dalal, Ajay Mandlekar, Caelan Reed Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. In Conference on Robot Learning, 2023. 3

[9] 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. 3

[10] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14953–14962, 2023. 3

[11] Xuehai He, Weixi Feng, Kaizhi Zheng, Yujie Lu, Wanrong Zhu, Jiachen Li, Yue Fan, Jianfeng Wang, Linjie Li, Zhengyuan Yang, et al. Mmworld: Towards multi-discipline multi-faceted world model evaluation in videos. In The Thirteenth International Conference on Learning Representations. 6

[12] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352, 3(4):6, 2023. 3

[13] Jen-tse Huang, Eric John Li, Man Ho Lam, Tian Liang, Wenxuan Wang, Youliang Yuan, Wenxiang Jiao, Xing Wang, Zhaopeng Tu, and Michael R Lyu. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. arXiv preprint arXiv:2403.11807, 2024. 3

[14] Yuzhou Huang, Yiran Qin, Shunlin Lu, Xintao Wang, Rui Huang, Ying Shan, and Ruimao Zhang. Story3d-agent: Exploring 3d storytelling visualization with large language models. arXiv preprint arXiv:2408.11801, 2024. 3

[15] 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. 3

[16] 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

[17] 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 Fortieth International Conference on Machine Learning, 2023. 3

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

[19] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. arXiv preprint arXiv:2406.09246, 2024. 2

[20] 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. 3

[21] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents

for" mind" exploration of large language model society. Advances in Neural Information Processing Systems, 36: 51991–52008, 2023. 3

[22] 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

[23] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. arXiv preprint arXiv:2408.06327, 2024. 6

[24] Teli Ma, Jiaming Zhou, Zifan Wang, Ronghe Qiu, and Junwei Liang. Contrastive imitation learning for language-guided multi-task robotic manipulation. arXiv preprint arXiv:2406.09738, 2024. 3

[25] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In Conference on Robot Learning, pages 879–893. PMLR, 2018. 3

[26] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. arXiv preprint arXiv:2003.06085, 2020. 3

[27] Manuel Mosquera, Juan Sebastian Pinzon, Manuel Rios, Yesid Fonseca, Luis Felipe Giraldo, Nicanor Quijano, and Ruben Manrique. Can llm-augmented autonomous agents cooperate?, an evaluation of their cooperative capabilities through melting pot. arXiv preprint arXiv:2403.11381, 2024. 3

[28] Yao Mu, Junting Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. arXiv preprint arXiv:2402.16117, 2024. 2, 3

[29] Yao Mu, Tianxing Chen, Shijia Peng, Zanxin Chen, Zeyu Gao, Yude Zou, Lunkai Lin, Zhiqiang Xie, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins (early version). arXiv preprint arXiv:2409.02920, 2024. 2, 3, 6

[30] Sanjay Nambiar, Marie Jonsson, and Mehdi Tarkian. Automation in unstructured production environments using isaac sim: A flexible framework for dynamic robot adaptability. Procedia CIRP, 130:837–846, 2024.

[31] 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. In Robotics: Science and Systems, 2024. 6

[32] Yiran Qin, Zhelun Shi, Jiwen Yu, Xijun Wang, Enshen Zhou, Lijun Li, Zhenfei Yin, Xihui Liu, Lu Sheng, Jing Shao, et al. Worldsimbench: Towards video generation models as world simulators. arXiv preprint arXiv:2410.18072, 2024. 3

[33] Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. Mp5:

A multi-modal open-ended embodied system in minecraft via active perception. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16307–16316, 2024. 3

[34] Yiran Qin, Ao Sun, Yuze Hong, Benyou Wang, and Ruimao Zhang. Navigatediff: Visual predictors are zero-shot navigation assistants. arXiv preprint arXiv:2502.13894, 2025. 3

[35] 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. 3, 6

[36] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023. 2

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017. 3

[38] Sai H. Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. IEEE Access, 12:55682–55696, 2024. 3

[39] David Venuto, Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and Ankit Anand. Code as reward: Empowering reinforcement learning with vlms. arXiv preprint arXiv:2402.04764, 2024. 3

[40] Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. Sibyl: Simple yet effective agent framework for complex real-world reasoning. arXiv preprint arXiv:2407.10718, 2024. 3

[41] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155, 2023. 3

[42] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 11097–11107, 2020. 6

[43] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In CVPR, 2024. 5

[44] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. arXiv:2406.09414, 2024. 5

[45] Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, et al. Oasis: Open agents social interaction simulations on one million agents. arXiv preprint arXiv:2411.11581, 2024. 3

[46] Xianhao Yu, Jiaqi Fu, Renjia Deng, and Wenjuan Han. Mineland: Simulating large-scale multi-agent interactions

with limited multimodal senses and physical needs. arXiv preprint arXiv:2403.19267, 2024. 3

[47] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In Proceedings of Robotics: Science and Systems (RSS), 2024. 3

[48] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In Proceedings of the AAAI Conference on Artificial Intelligence, pages 19632–19642, 2024. 3

[49] 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. 2, 3

[50] Enshen Zhou, Yiran Qin, Zhenfei Yin, Yuzhou Huang, Ruimao Zhang, Lu Sheng, Yu Qiao, and Jing Shao. Minedreamer: Learning to follow instructions via chain-of-imagination for simulated-world control. arXiv preprint arXiv:2403.12037, 2024. 3

[51] Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. arXiv preprint arXiv:2412.04455, 2024. 3

[52] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. arXiv preprint arXiv:2307.13854, 2023. 3

RoboFactory: Exploring Embodied Agent Collaboration with Compositional Constraints

Supplementary Material

A. Data Generation

In this section, we provide a detailed description of the process for effectively generating expert data. First, we elaborate on the details of RoboBrain, explaining how it generates the next subgoal and constraints. Next, we introduce the method for generating agent trajectories based on subgoals and constraints. Then, we describe the implementation of RoboChecker, an interface to integrate various constraints with data generation pipeline. Finally, we present tasks in the *RoboFactory* benchmark along with its corresponding descriptions.

RoboBrain In RoboBrain, we structure the following prompts for GPT-4o [1] to generate new subgoals based on the given information, such as task instructions, previous subgoals, and constraint violation feedback. Along with each subgoal, a set of constraints is generated, which can be categorized into three levels: logical, temporal, and spatial. These constraints are formulated as structured text to ensure that RoboChecker can accurately recognize the corresponding functions and verify whether the constraints are satisfied. The detailed description of prompts is as follows.

```
You are an AI system responsible for generating subgoals and constraints for a multi-agent robotic task.
 Your goal is to ensure that each agent receives a clearly defined subgoal while adhering to
 well-structured constraints. Constraints must be formatted correctly to enable validation and enforce
 coordination and collaboration among agents.

Key Requirements
- Generate at least one subgoal per agent based on the given task description.
- Define explicit constraints for each agent, ensuring every constraint involves at least one agent.
- Follow specific formatting rules to categorize constraints accurately.
- Ensure all constraints are clear, actionable, and unambiguous to guide robotic agents effectively.

Input Structure
- Task Instruction: "{General description of the task}"
- Global Observation: <image_global>
- Agents Observation: [<image_1>,<image_2>....,<image_n>]
- Previous Subgoals: "{Subgoals executed by each agent}"
- Constraint Violation Feedback: "{List of feedback from violated constraints, if any}"

Output Content
- A set of subgoals and constraints based on the task requirements. Each constraint should follow these
 formatting rules according to its category:
1. Logical Constraints:
   - Agent-specific condition: Agent-Specific Condition: Specifies a requirement for the behavior of a
 single agent.
      Example: "The gripper of Agent_1 must be perpendicular to {Object}."
   - Multi-agent condition: Defines a coordination rule between multiple agents.
      Example: "Agent_1 and Agent_2 must maintain a consistent gripper height.".
2. Temporal Constraints:
   - Synchronization: Specifies whether agents can perform tasks simultaneously or share the same space.
      Example: "Agent_1 and Agent_3 perform tasks simultaneously without interference.".
   - Sequence: Defines the required order of actions between agents.
      Example: "Agent_2 must complete the task before Agent_4 can begin their action.".
3. Spatial Constraints:
   - Collision avoidance: Ensures agents do not interfere with each other or the environment.
      Example: "Agents must avoid colliding with each other when moving in close proximity.".
   - Space occupancy: Specifies spatial positioning rules to prevent conflicts.
      Example: "Agent_1 should not occupy the same space as Agent_3 in the designated area.".

Output Example
{
  "Subgoals": {
    "Agent_1": "{Clear and structured subgoals for Agent_1}",
    "Agent_2": "{Clear and structured subgoals for Agent_2}",
    ...
  },
  "Constraints": {
    "Logical": [
      {
```

```
          "Agent": "Agent_1",
          "Constraint": "The gripper of Agent_1 is perpendicular to {Object}."
        },
        {
          "Agents": ["Agent_2", "Agent_3"],
          "Constraint": "Keep the gripper height consistent between Agent_2 and Agent_3 to make the camera
  remain horizontal."
        }
      ],
      "Temporal": [
        {
          "Agents": ["Agent_2", "Agent_4"],
          "Constraint": Agent_2 and Agent_4 could share the same space chronologically."
        }
      ],
      "Spatial": [
        {
          "Agent": "Agent_2",
          "Constraint": "Avoid collision between Agent_2 and other Agents."
        },
        {
          "Agent": "Agent_4",
          "Constraint": "Avoid collision between Agent_4 and other Agents."
        }
      ]
    }
  }
```

Trajectory Generation   Effectively converting these conceptual subgoals into precise robotic motion trajectories remains a significant challenge for large language models. Inspired by RoboTwin [29], we define a set of motion primitives, each represented as a Python function interface. By providing specific input parameters, these primitives generate corresponding motion trajectories. For instance, the MOVE primitive inputs an agent ID and a target position. Then, it computes a trajectory based on the current position of the robotic arm to generate the appropriate motion sequence. This approach allows large language models to focus on understanding the high-level logic of action interactions while avoiding direct involvement in low-level control signal computations.

RoboChecker   RoboChecker is designed to evaluate the validity and efficiency of generated motion trajectories, ensuring smooth execution while preventing collisions and inconsistencies. To achieve this, we define four key validation functions as optional interface type, each addressing a specific aspect of trajectory assessment: agent movement direction, interaction at contact points, spatial occupancy of trajectories, and the correctness of trajectory scheduling. The definitions of these functions are as follows:

- **Movement Direction Validation**: Ensures that the movement of each agent aligns with logical constraints. For instance, a robotic gripper should maintain an appropriate angle when grasping an object or adhere to a specific orientation during task execution to guarantee stable and effective interactions.
- **Contact Point Interaction Validation**: Verifies whether an interaction with object or other agents meets expected conditions. In collaborative manipulation tasks, for example, multiple agents should grasp objects at appropriate positions to maintain stability during joint handling.
- **Spatial Occupancy Validation**: Analyzes the spatial feasibility of an agent's trajectory, ensuring that it does not enter restricted zones or cause spatial conflicts. For instance, in confined environments, different agents' paths should remain non-overlapping to avoid collisions.
- **Trajectory Scheduling Validation**: Assesses whether the execution order of motion trajectories adheres to temporal constraints. This includes ensuring that actions requiring synchronization occur simultaneously and that tasks with sequential dependencies are executed in the correct order. It will also analyze whether these operations can be executed simultaneously or follow a predefined sequence along the trajectory. For example, in a task where a lid must be opened before placing an object inside, the action of "open lid" should precede the action of "place object" in the trajectory plan.

These functions take two types of inputs: the current agent's trajectory and the constraints generated by RoboBrain. The constraints produced by RoboBrain are represented in textual form. To process these constraints effectively, we construct the following prompt to match each constraint with its corresponding function, extract the relevant parameters from the constraint text, and integrate them with the agent's trajectory for validation. In Fig 6, we present the constraints of the Take Photo task along with the CheckCode generated through visual programming based on these constraint, which serves as the evaluation

protocol, bridging textual constraints with the corresponding trajectory.

```
You are an expert in robotic motion validation, responsible for ensuring that a given set of motion
 trajectories adheres to logical, spatial, and temporal constraints. Your task is to validate these
 trajectories based on predefined requirements, ensuring compliance with movement logic, spatial
 integrity, and execution order.
To achieve accurate validation, you must:
- Match each constraint to the appropriate validation function.
- Extract relevant parameters from the constraint description.
- Apply the corresponding validation rule to assess compliance.

Validation Functions and Parameter Extraction
Each constraint is assigned to a specific validation function, which extracts relevant parameters and
 applies the appropriate validation rule.
1. Movement Direction Validation: Ensures that an agent maintains the required orientation during
 interactions.
    Extracted Parameters:
        Agent ID: The agent executing the movement.
        Target Object: The object involved in the interaction.
        Required Orientation: The necessary orientation for the gripper of the agent.
    Formal Representation:
        (Agent_ID, Target_Object, Required_Orientation) -> Validate_Direction()
    Example Constraint:
        "The gripper of Agent_1 must be perpendicular to Object_A when grasping."
        (Agent_1, Object_A, perpendicular) -> Validate_Direction()
2. Contact Point Interaction Validation: Ensures that agents interact with objects or other agents at the
 designated contact points.
    Extracted Parameters:
        Agent ID: The agent performing the interaction.
        Target Object: The Object involved in the interaction.
        Contact Point: The designated interaction point (\eg, left side of the object).
    Formal Representation:
        (Agent_ID, Target_Object, Contact_Point) -> Validate_Interaction()
    Example Constraint:
        "Agent_3 must grasp Object_B at its left point."
        (Agent_3, Object_B, left) -> Validate_Interaction()
3. Spatial Occupancy Validation: Ensures that the movement of an agent does not result in spatial
 conflicts, such as entering restricted zones or colliding with other agents.
    Extracted Parameters:
        Agent IDs: The agents whose trajectories require validation.
    Formal Representation:
        (Agent_IDs) -> Validate_Spatial_Occupancy()
    Example Constraints and its parameters:
        "Agent_2 must not intersect with the trajectories of other agents."
        (Agent_2) -> Validate_Spatial_Occupancy()
4. Trajectory Scheduling Validation: Ensures that the execution order of actions adheres to temporal
 constraints,  including:
    - Sequential dependencies, where one action must precede another.
    - Synchronized execution, where multiple agents must act simultaneously.
    Extracted Parameters:
        Agent IDs: The agents involved in the scheduling constraint.
        Task Dependency Type:
            - Sequential: Specifies an ordered execution sequence.
            - Simultaneous: Requires two agents to perform actions at the same time.
    Formal Representation:
        (Agent_IDs, Task_Dependency_Type) -> Validate_Scheduling()
    Example Constraints and its parameters:
        "Agent_4 must place Object_C only after Agent_5 opens the container."
        ([Agent_5, Agent_4], "Sequential") -> Validate_Scheduling()
```

Tasks   Our benchmark dataset includes 11 tasks. Table 6 presents the number of agents for each task, the task description, and the corresponding target condition. While single-agent tasks can assess the robotic arm's interaction capabilities, our primary focus is on multi-agent tasks, which evaluate the coordination and cooperation abilities between agents.

B. Experimental Setup

B.1. Training Details

We adopt the CNN-based Diffusion Policy as our base model, with a prediction horizon of 8, observation steps are set to 3, and action steps are set to 6. For the dataloader, we use a batch size of 128. The optimizer is set to `torch.optim.AdamW` with a learning rate of 1.0×10^{-4}, betas in the range of $[0.95, 0.999]$, and ϵ set to 1.0×10^{-8}. The learning rate warmup lasts

500 steps, and we train for 300 epochs for all tasks in the benchmark. The training process is conducted on a single Nvidia RTX 4090 GPU. For 150 demonstration samples with average episode length of 205, the training time is around 5 hours.

Different Training Strategies   Each Franka Emika Panda robotic arm consists of seven rotational joints, with an additional one-dimensional action for the gripper (as both left and right grippers maintain the same width), resulting in an action space of dimension 8 per agent. We design four different multi-agent DP strategy modes:

- Global View and Shared Policy: All agents share a global observation that includes every agent in the environment. For a task with N agents, their actions are concatenated to form a joint action of dimension $8N$. A single model is trained using the global view and the $8N$-dimensional joint action.
- Local View and Shared Policy: Each agent has an individual local observation centered on its own action. To prevent catastrophic forgetting during training, we randomly shuffle the training data of all agents before inputting them into a shared model. A single model is trained with multiple local observations and the corresponding agent's 8-dimensional action.
- Global View and Separate Policy: All agents share a global observation, ensuring that all agents are included within it. However, each agent trains its own model to determine actions independently. The training input consists of the global view and an individual agent's 8-dimensional action.
- Local View and Separate Policy: Each agent has an individual local observation centered on its own action, with its own perspective prioritized while incorporating surrounding environmental information. Each agent trains a separate model using its local view and corresponding 8-dimensional action.

All global and local views used for training are RGB images with a resolution of 320×240. We select the fourth training strategies as the baselines for the benchmark experiments.

B.2. Evaluation Details

To ensure smooth robotic arm movements during simulations, we employ interpolating operation for action trajectories generated by the Diffusion Policy. For each task, we conduct evaluations across 100 distinct scene configurations, varying initial object placements and environmental conditions. We introduce a maximum action step limit for each task for evaluation of success rates. A failure is determined if the task is not completed within this limit. To set a reasonable threshold, we perform a warm-up test among 20 samples to estimate the average number of steps required to complete the task. The maximum action step limit is set twice this average. The success criteria for each task, including the target conditions, are detailed in Table 6.

C. Demonstrations of Benchmark

Fig. 7 demonstrates several tasks in the *RoboFactory* benchmark. We visualize the observation of key timestamps with the corresponding subgoals generated from RoboBrain across 4 tasks (*Camera Alignment*, *Place Food*, *Two Robot Stack Cube*, *Lift Barrier*).

Goal: Grab the *steak* and use the *camera* to *photograph* it with 4 Embodied Agent.

Observation

Subgoals

Agent1: Grasp the steak.
Agent2: Grasp the camera handle1.
Agent3: Grasp the camera handle2.
Agent4: Press the camera shutter.

Agent1: Lift the steak align the camera.
Agent2: Lift the camera handle1.
Agent3: Lift the camera handle2.
Agent4: Press the camera shutter.

Agent1: Stay still.
Agent2: Stay still.
Agent3: Stay still.
Agent4: Press the camera shutter.

Constraints

Logical:
1. **Agent1's** gripper is perpendicular to the steak.
2. **Agent2** and **Agent3** should interact with the camera left and right handle point
Spatial:
1. Avoid collision between **Agent2** and other Agents.
2. Avoid collision between **Agent3** and other Agents.
Temporal:
1. **Agent1**, **Agent2** and **Agent3** should move at the same time
2. **Agent4** should move after **Agent1**, **Agent2** and **Agent3.**

Logical: None.
Spatial:
1. Avoid collision between **Agent1** and other Agents.
2. Avoid collision between **Agent2** and other Agents.
3. Avoid collision between **Agent3** and other Agents.
Temporal:
1. **Agent1**, **Agent2** and **Agent3** should move at the same time
2. **Agent4** should move after **Agent1**, **Agent2** and **Agent3.**

Logical:
1. **Agent4** should interact with the camera's shutter contact point.
2. **Agent4's** gripper is perpendicular to the camera.
Spatial:
1. Avoid collision between **Agent4** and other Agents.
Temporal: None.

CheckCode

VI(Agent2, camera, "handle1")
VI(Agent3, camera, "handle2")
VD(Agent1, steak, "perpendicular")
VSO(Agent2)
VSO(Agent3)
VS([Agent1, **Agent4**], "Sequential")
VS([Agent2, **Agent4**], "Sequential")
VS([Agent3, **Agent4**], "Sequential")
VS([Agent1, **Agent2**, **Agent3**], "Simultaneous")

VSO(Agent1)
VSO(Agent2)
VSO(Agent3)
VS([Agent1, **Agent4**], "Sequential")
VS([Agent2, **Agent4**], "Sequential")
VS([Agent3, **Agent4**], "Sequential")
VS([Agent1, **Agent2**, **Agent3**], "Simultaneous")

VI(Agent4, camera, "shutter")
VD(Agent4, camera, "perpendicular")
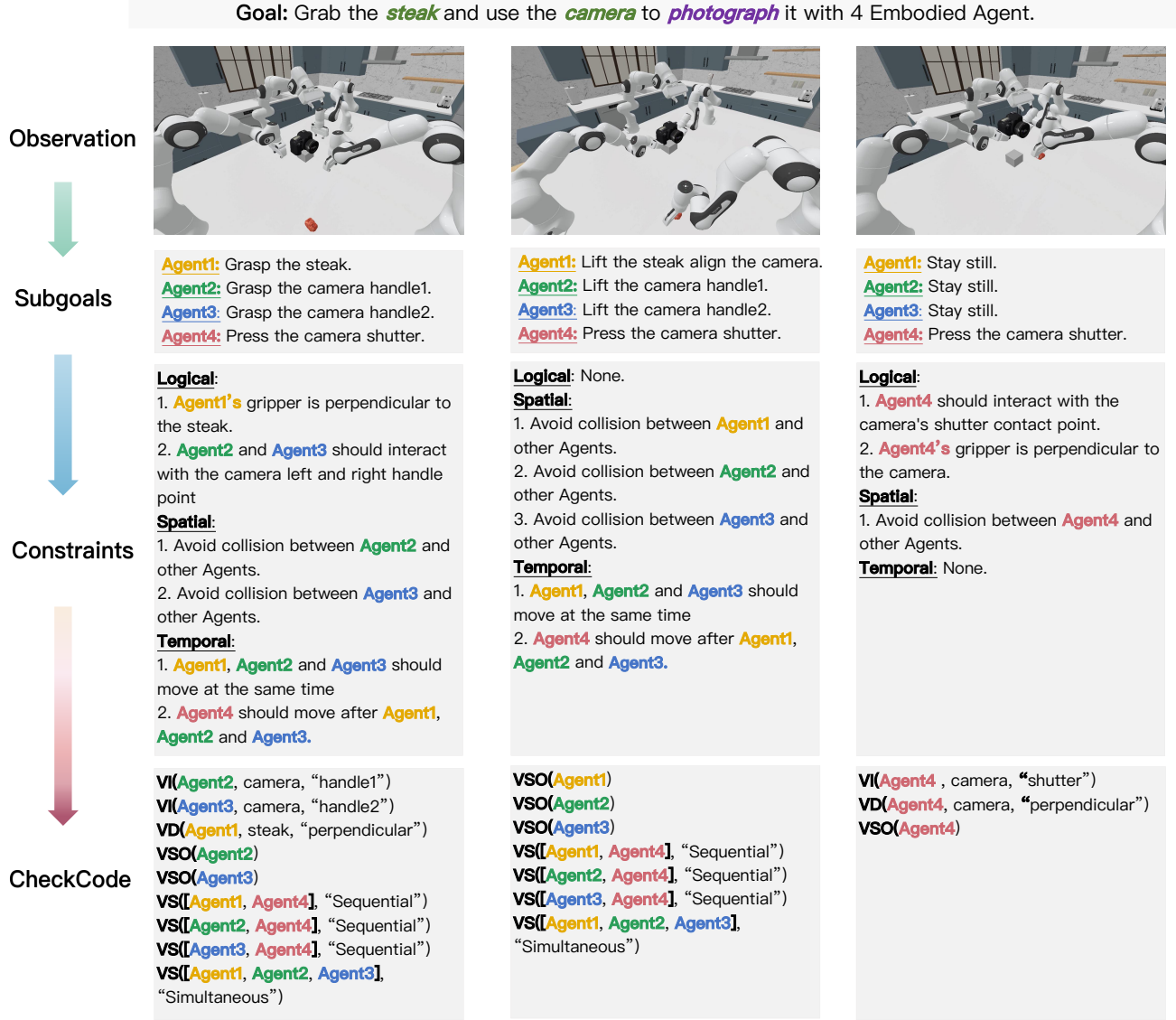**VSO(Agent4)**

Figure 6. Demonstration of *RoboChecker* is showcased in the complete execution of the Take Photo task. By analyzing constraints, *RoboChecker* generates **CheckCode**, a composition of multiple interfaces. Specifically, VI stands for **Validate Interaction**, VD for **Validate Direction**, VSO for **Validate Spatial Occupancy**, and VS for **Validate Scheduling**. The CheckCode returns true only when all interfaces pass validation, indicating that the generated motion trajectory adheres to the compositional constraints. Otherwise, CheckCode identifies the failed interfaces and sends the feedback to *RoboBrain*.
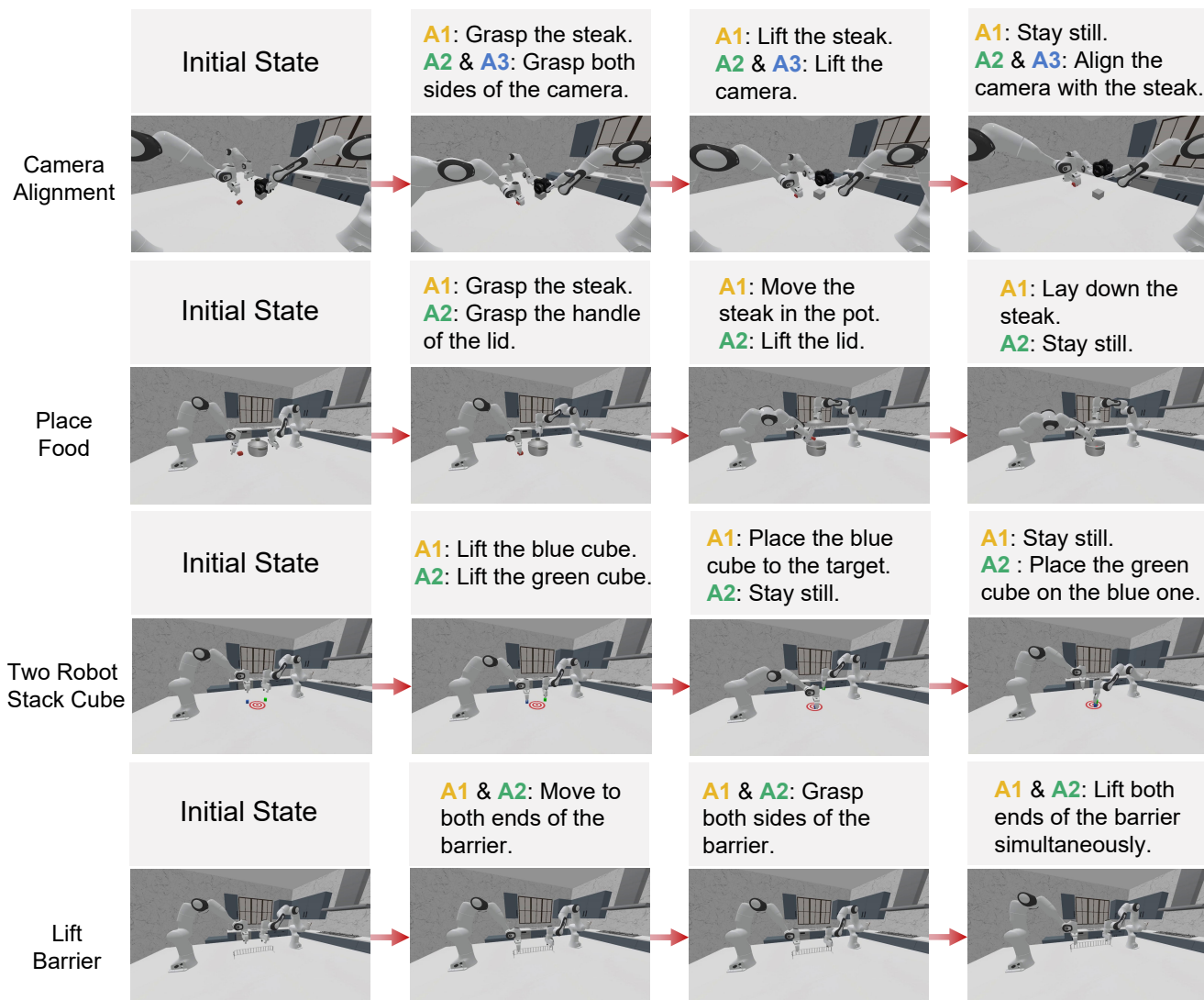
Figure 7. Demonstrations of tasks in the *RoboFactory* Benchmark. For each task, the subgoals in each timestamp are displayed in the top row, and the observation is shown in the bottom row.

Task	Agent Number	Description	Target Condition
Pick Meat	1	There is a piece of meat placed on the table. A robotic arm picks up the meat and lifts it to a specified height.	The height of the meat reaches a predefined threshold.
Stack Cube	1	A blue cube and a red cube are placed on the table. A robotic arm picks up the blue cube and places it on top of the red cube.	The distance between the blue and red cubes is within a threshold, with the blue cube positioned at a greater height than the red cube.
Strike Cube	1	A hammer and a cube are placed on the table. A robotic arm first identifies an optimal grasping position to pick up the hammer, then moves it to a suitable position to strike the cube.	The hammerhead is positioned directly above the cube within a predefined distance threshold.
Lift Barrier	2	A long barrier is placed on the table. Two robotic arms simultaneously grasp both ends of the barrier and lift it to a specified height.	The barrier is elevated to the specified height while maintaining stability.
Pass Shoe	2	A shoe is placed on the table. One robotic arm grasps the shoe and passes it to the other one, which then places it at the target location.	The distance between the shoe and the target location is within a predefined threshold.
Place Food	2	A pot and a kind of food are placed on the table. One robotic arm lifts the pot's lid, while the other picks up the food and places it inside the pot.	The food is placed inside the pot, with the distance between the food and the center of the pot being within a predefined threshold.
Two Robots Stack Cube	2	A blue cube and a red cube are placed on the table. A robotic arm picks up the blue cube to a specified position, while the other places the red cube on top of it.	The blue cube is within the specified threshold distance from the target position. The distance between the blue and red cubes remains within a defined threshold, with the red cube positioned at a greater height than the blue cube.
Camera Alignment	3	A camera and an object are placed on the table. One robotic arm picks up the object to a specified position. The other two robotic arms grasp both sides of the camera and align it to the object.	The camera reaches a specified height, and the object is placed at the designated position that aligns with the camera.
Three Robots Stack Cube	3	A blue cube, a red cube and a green cube are placed on the table. One robotic arm picks up the blue cube to a specified position. Another arm places the red cube on top of the blue one. The last arm places the green cube on top of the red one.	The blue cube is positioned within the specified target range. Additionally, the red cube is successfully placed on top of the blue cube, and the green cube is positioned atop the red cube.
Take Photo	4	A camera and an object are placed on the table. One robotic arm picks up the object and places it to a specified position. Another two robotic arms grasp both sides of the camera and align it to the object. The last robotic arm clicks the shutter.	The camera reaches a specified height, and the object is placed at the designated position that aligns with the camera. Additionally, the distance between the end effector of the last robotic arm and the camera's shutter is within a certain threshold.
Long Pipeline Delivery	4	A shoe is placed on the table. Three robotic arms grasp the shoe and pass it to the next robotic arm. The last robotic arm places the shoe to a specified position.	The distance between the shoe and the target location is within a predefined threshold.

Table 6. Task Descriptions for the *RoboFactory* Benchmark