https://doi.org/10.26599/AIR.2024.9150037 CAAI Artificial Intelligence Research

# Growing from Exploration: A Self-Exploring Framework for Robots Based on Foundation Models

Shoujie Li[1], Ran Yu[1], Tong Wu[1], Junwen Zhong[2], Xiao-Ping Zhang[1], and Wenbo Ding[1, 3] ✉

## ABSTRACT

Intelligent robot is the ultimate goal in the robotics field. Existing works leverage learning-based or optimization-based methods to accomplish human-defined tasks. However, the challenge of enabling robots to explore various environments autonomously remains unresolved. In this work, we propose a framework named GExp, which endows robots with the capability of exploring and learning autonomously without human intervention. To achieve this goal, we devise modules including self-exploration, knowledge-base-building, and close-loop feedback based on foundation models. Inspired by the way that infants interact with the world, GExp encourages robots to understand and explore the environment with a series of self-generated tasks. During the process of exploration, the robot will acquire skills from experiences that are useful in the future. GExp provides robots with the ability to solve complex tasks through self-exploration. GExp work is independent of prior interactive knowledge and human intervention, allowing it to adapt directly to different scenarios, unlike previous studies that provided in-context examples as few-shot learning. In addition, we propose a workflow of deploying the real-world robot system with self-learned skills as an embodied assistant. Project website: GExp.com.

## KEYWORDS

intelligent robot; foundation models; self-exploring framework

Not only for humans[1, 2], autonomous exploration and learning are equally important for robots[3], which is a sign of robot intelligence[4]. Autonomous robots can not only liberate people from hard or dangerous labor but also complete some daily and ubiquitous tasks, such as cloth-folding[5–7], cooking[8, 9], and cleaning[10–12]. However, the design of robots with autonomy is very challenging. While robots can complete some tasks with the help of reinforcement learning[13], imitation learning[14], and other methods, they are only capable of dealing with related tasks and cannot learn by themselves. Despite these methods have achieved outstanding results, they constrain the automatic learning ability of robots and it is difficult to achieve real intelligence only with these methods.

Recently, the birth of Large Language Models (LLMs), such as GPT-4[15] and Llama 2[16], provides new paths to intelligence. LLMs have not only propelled the development of artificial intelligence technology but have also significantly enhanced the decision-making and task-planning capabilities of robots. There is a growing body of research utilizing LLMs to address existing robotics challenges, as evidenced in previous studies[17–19]. However, most existing methods rely heavily on detailed, complete prior information and the provision of human-specific in-context examples. This dependency often limits the robots' generalization capabilities across diverse environments.

On the other hand, there are studies demonstrating the potential of creating autonomous agents based on LLMs[7, 20], which are capable of independently and continuously resolving problems[21, 22]. A notable example is Voyager[22], which presents an embodied agent operating within a Minecraft virtual environment. Nevertheless, these agents, built upon LLMs, interact with their environment through textual feedback, which hinders practical applications in robotics. The recent advancements in vision-language models have been particularly significant, showcasing the potential for general object recognition and understanding of spatial relationships[23, 24]. This progression opens up new possibilities for robotics, allowing for more nuanced and adaptable interactions with various environments.

To facilitate autonomous exploration and learning in robots, we introduce an innovative framework named GExp, which is predicated on the use of foundational models. As shown in Fig. 1, the framework is inspired by the growth process of infants. Newborn babies begin with an incomplete understanding of the world's logic, gradually perceiving the world through their vision and interacting with it physically, leading to progressive development. Similarly, in GExp, pre-trained foundational models are employed to assist the robot in navigating and learning within unfamiliar environments. With an understanding of the environment with vision-language model (VLM), robots will generate tasks and try to solve them with the ability of LLM. Throughout the exploration phase, the robot will learn from successful experiences by generating relevant skills in a zero-shot manner. These skills are continuously collected into a comprehensive library, allowing the robot to tackle increasingly complex scenarios.

1 Shenzhen Ubiquitous Data Enabling Key Lab, Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China
2 Department of Physics and Chemistry, Faculty of Science and Technology, University of Macau, Macau 999078, China
3 RISC-V International Open Source Laboratory, Tsinghua-Berkeley Shenzhen Institute, Shenzhen 518055, China
Shoujie Li, Ran Yu, and Tong Wu contribute equally to this work.
Address correspondence to Wenbo Ding, ding.wenbo@sz.tsinghua.edu.cn

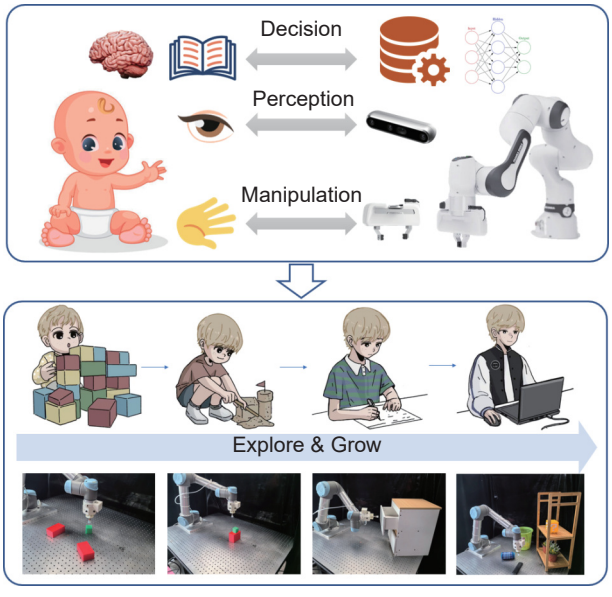


**Fig. 1 An infant-inspired framework for autonomous robotic exploration (From left to right: pick blocks, stack blocks, open drawer, and classify objects).**

The contribution of the article mainly includes:

- A framework for robot autonomous self-exploration is proposed, which does not require the imposition of specific human-defined tasks, nor does it necessitate any prior knowledge about the environment. Its primary function is to facilitate continuous and independent exploration by the robot.
- In our approach, we enable robots to learn during exploration by leveraging successful experiences. This is achieved through the use of LLM, which assists in generating general skills in a zero-shot manner. The skills thus developed not only enhance the robot's capacity to solve increasingly complex tasks encountered during exploration but also expand the boundaries of its abilities.
- We create a self-verification module using a pre-trained VLM to analyze and determine task execution success. This evolves into "backtracking control" enabling the robot to verify each task step's success by assessing preconditions. This ensures actions align with the overall task objective, enhancing the robot's precision and effectiveness.
- To evaluate the exploration and self-learning capabilities of the robot effectively, we conduct a series of experiments specifically designed to validate the feasibility and effectiveness of our proposed framework.

## 1 Related Work

**Foundation models for robotics.** The evolution of LLMs has significantly transformed the robotics field, particularly in areas such as task planning[25, 26], robot action generation[27], and code development[18, 28]. These models have proven effective in integrating with traditional motion planning algorithms[17] or reinforcement learning[29], enhancing capabilities in multi-modal information processing, multi-robot coordination, and human-robot collaboration. Another kind of foundation model, VLMs[25, 30, 31], also fast developed recently and many researches have been conducted to discuss its impact in robotics[23]. Our work combines the LLM and VLM to construct an autonomous embodied agent with the ability to explore the real world.

**Language models for task and motion planning.** Long-horizon planning problem is one of the key research areas in robotics[32]. Conventional methods are mainly based on optimization and reinforcement learning methods[33]. Recently, some researchers utilized LLM as the core of robots to divide sub-tasks or generate high-level plans for task and motion planning[19, 26, 28, 34]. To help the LLM better understand the given task and reduce the hallucinations in generating plans, in-context examples are used in most of the relevant works. However, this could reduce the generalization ability, since those examples should be collected manually for a given environment. We are trying to improve the generalization ability of LLM-based planning problems by designing a framework for robot self-exploration.

## 2 Method

### 2.1 Grow from exploration

Assuming a robot faces an unfamiliar environment, it is not provided with any human-specific tasks and prior knowledge about the environment. The robot can observe the environment with an RGB-D camera and interact with the objects in the environment with a robot arm. Our goal is to let robots autonomously interact with the environment to explore the boundaries of their abilities. To reach that goal, a robot should understand the environment, generate tasks by itself, make plans, and interact with the environment to find out the feasibility of generated plans (see Fig. 2).

#### 2.1.1 Scene understanding and tasks generation

In our setting, the robot does not have access to any prior information about the environment. Therefore, robots need to first perceive and understand their surroundings. We leverage VLMs to characterize the information in the scene with an image as input. The purpose of scene understanding can be divided into two parts: identifying observed objects with their attributes and generating relation descriptions that explain the relative positions of different objects.

After obtaining the scene description and information about the objects in the space, the robot will ask LLM to generate a series of manipulation tasks to explore the environment and possibly feasible skills. A task is characterized by its name, related objects, and a detailed description. To ensure efficient task generation, we employ an object-related principle. This principle stipulates that each generated task must contain at least one observed object, preventing LLM from generating some useless tasks. Inspired by the growth of infants and curriculum learning, we also prompt LLM to generate structured tasks from simple to complex. Simpler tasks typically involve operations on individual objects and are characterized by straightforward descriptions and a limited number of steps. Complex tasks involve multiple objects and normally relate to long-horizon planning.

#### 2.1.2 Planning and executing

LLMs have strong reasoning and planning abilities. It has also been proved that LLMs are capable of grounding a high-level plan into several sub-actions. Motivated by these, we developed a module that uses LLM as a planner and controller. LLM is given a self-generated task and a library of acquired skills and outputs a high-level plan with an action sequence that is composed of the acquired skills. These skills can be in any form, such as trained policy through reinforcement learning and imitation learning, or just code. In this paper, we choose code as the form of skills for simplicity. To improve the reasoning ability of LLM, we utilize

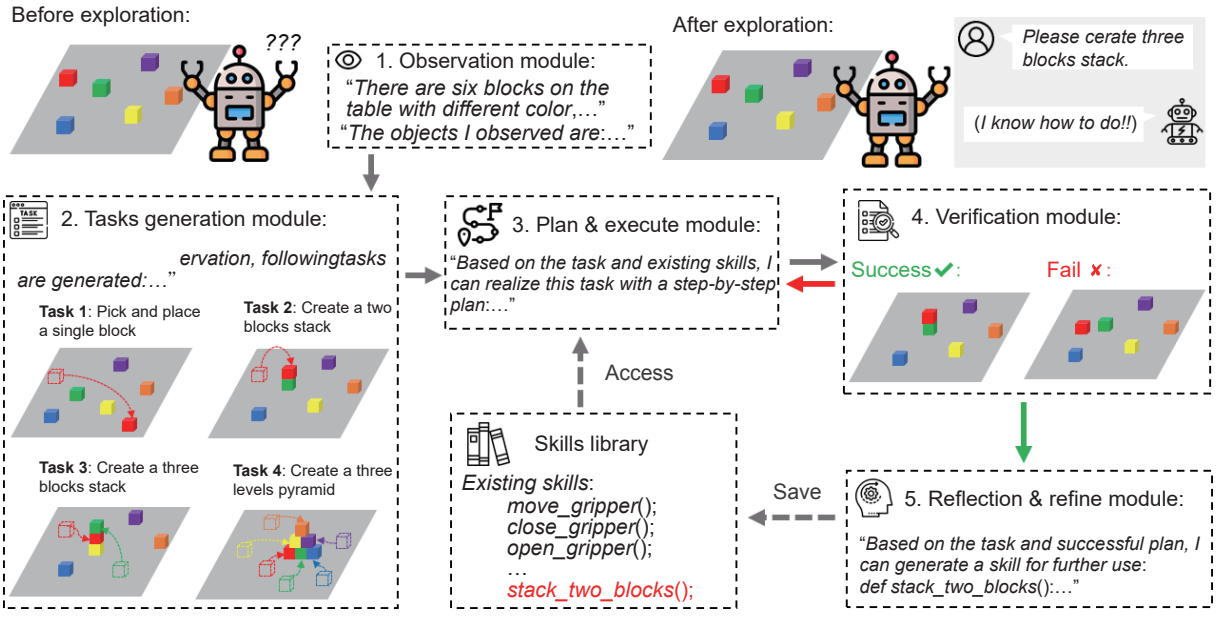


**Fig. 2 Robot exploration framework GExp. GExp allows robots to actively explore the new environment by observing the environment, generating feasible tasks, solving those tasks independently, and verifying the success of execution. During the process of exploration, GExp maintains and updates a skills library by reflecting on successful tasks and generating task-related skills.**

Chain-of-Thought (CoT)[35] in the process of plan generation. In this paper, we choose code as the form of skills, as code can be directly generated by LLM without extra training or human demonstrations. However, using code for grounding has limitations in complex motion planning, which is not the main focus of our work.

During the exploration, the robot will also maintain a skill library that contains all acquired skills. The skill library is kept updated by LLM generating a new one when the task is completed. The following functions are predefined in the initial skill library as primitive functions:

- movep(position): Move the end-effector of the robot to a given position.
- close_gripper(): Close the gripper to grasp an object. The gripper is initially open.
- open_gripper(): Open the gripper to place an object.
- get_obj_position(object_name): Get the position of an object with the given name. The output of this function is a tuple of object position $(x, y, z)$.
- go_home(): Move the gripper to the home position.

The choosing principles of those are generalization and simplicity. From the perspective of generalization, those skills should be required by most manipulation tasks and can be easily defined for any robot system. From the perspective of simplicity, we want to inspire robots to generate more high-level and task-specific skills based on those simple skills, instead of heavily relying on human-specific actions. Except for those functions, we also define the robot arm's available working space for safety reasons:

- BOUNDS: The available working space in three dimensions, $[[x_{min}, x_{max}], [y_{min}, y_{max}], [z_{min}, z_{max}]]$.

### 2.1.3 Verification and error correction

After implementing the generated plan, it is crucial to verify its success. The GExp operates autonomously based on foundation models, thus excluding any human involvement in confirming the success of the implemented manipulation plan. To ascertain whether the task has been completed successfully, foundation models serve as success detectors.

We utilize two different methods for self-verification: code-based method and vision-based method. The former one, utilizing the reasoning and code-creating ability of LLM, generates a success detection function based on the description of the given task. The vision-based method leverages advanced VLMs (GPT-4V) to judge whether the task requirement is satisfied. It is worth mentioning that existing VLMs have limited capability in object detecting, semantic grounding, and relation understanding. To improve its performance, we provide VLM with a final state observation image and initial state description (generated at the scene understanding stage). This can significantly help VLM make correct answers in success verification.

In the event of a plan failure due to errors, GExp will attempt to solve them. Errors can be divided into two categories: interpretation errors and grounding errors. Interpretation errors occur during the code generation phase and include syntax errors, which breach programming language rules, and invalid instance errors, characterized by the use of undefined variables or functions. The grounding errors are associated with the execution phase of the code. Common grounding errors include manipulating with object in a gripper and directional misjudgments.

Our approach to addressing these two types of errors differs. For interpretation errors, we solve them by letting LLM regenerate the code without changing the plan. This is achieved by providing the LLM with the error code with a corresponding error message. Results show that in most cases, these interpretation errors can be rectified in a single iteration. The second type of error, grounding error, can be detected by self-verification but hard be corrected by LLM itself without human intervention. We take an iterative plan-exploration strategy: regenerating the high-level plan by modifying the existing one. LLM is required to modify the current plan. However, it should be mentioned that not all self-generated plans are feasible to solve or exceed the current capabilities of foundation models.

### 2.1.4 Reflection and skill acquirement

If the success of the task has been verified, LLM is required to reflect the generated plan and code to refine a skill function with

the following rules:

• The created function should be general, which makes sure the function can be more likely to be used in further tasks.

• The LLM should generate the description of the new function, including its usage, input, and output.

• The current task should be resolved by using a new function, which serves as an example for further utilization.

Once the function has been generated, it should be first checked with no syntax error and then updated to the skills library.

### 2.1.5 Prompting design

The following rules are followed when designing the prompt:

• Format regularization: To ensure the stability of the system, all outputs of LLM follow the JSON format. This is realized by providing a format example in the system prompt.

• Prompt iteration: When the robot generates a new skill function from the interaction with the environment, the prompt should iterated to keep consistent with the current skills library.

• Generalized prompt: As one of the main motivations, we do not include any prior information about the environment and in-context examples. The framework can be applied to different environments without modifying the prompt.

## 2.2 How acquiring skills strengthen robot?

Foundation models have been pre-trained with tons of data from the Internet. However, when they are used for planning complex real-world problems, their performances are influenced by hallucinations. The performance can be improved by providing in-context examples by humans, while we want the robot to enhance its ability by learning from experiences in exploration (see Algorithm 1).

Denote the set of manipulation tasks can be performed by combining provided primary functions is $S^*_{\Pi_0} = \{T^*_1, T^*_2, \ldots, T^*_n\}$, the probability that LLM successfully finish the task $T^*_i$ is $P_{\text{LLM}}(T^*_i|\Pi_0)$. The goal of self-exploration is enabling the robot to automatically learn several skills $\Pi$ during the process of interaction between real-world environments to make $\sum_i P_{\text{LLM}}(T^*_i|\Pi) > \sum_i P_{\text{LLM}}(T^*_i|\Pi_0)$.

As introduced in the previous sections, the skill-collecting module can facilitate the exploration process. The robot can use previously generated skills to solve more complex problems. The generated skills form a tree structure: any new skill can be realized by the combination of basic skills (functions), and generated skills are also interrelated to each other. When the creation of skills is finished after one or multiple explorations, the skills library can be edited manually by deleting redundant skills or further distilled with LLM when combined with other skills libraries.

**Algorithm 1 Learn skills through exploration**

Input: Initial visual observation $x_0$, initial skills library with basic actions $\Pi = \Pi_0$, scene language description $l$, observed objects $O_{1:j}$

Output: Refined skills library $\Pi$

$l, O_{1:j} = \text{VLM}(x_0)$ ▷ See Appendix A.1
$T_{1:k} = \text{LLM}(l, O_{1:j})$ ▷ See Appendix A.2
**for** $T_i$ in $T_1, T_2, \ldots, T_k$ **do**
  $T = T_i, t = 1$
  **while** $t <$ num_retries **do**
    $p, a_{1:N} = \text{LLM}(T, \Pi)$ ▷ See Appendix A.3
    Execute plan $p$ with actions $a_{1:N}$
    flag_success = $\text{VLM}(x_i)$ ▷ See Appendix A.4
    **if** flag_success **then**
      $\pi = \text{LLM}(p, a_{1:N}, \Pi)$ ▷ See Appendix A.5
      $\Pi = \Pi \cup \{\pi\}$
      break
    **else**
      $t = t + 1$
    **end if**
  **end while**
**end for**
**return** Refined skills library $\Pi$

## 2.3 Deployment after exploration with backtracking control

After the robot explores one or more scenes, it is equipped with a self-learned skills library and able to complete user-given tasks within its capabilities. Different from exploring with self-generated tasks, the robot is provided with free-from-language instruction, such as "Clean the table" and "Where is my cup".

To deploy the robot after exploration, we utilize the workflow shown in Fig. 3. Inspired by the idea of ReAct[36], the robot is controlled by a controller powered by LLM which will continuously switch between observer module and executor module, until satisfying the user's requirement. When receiving user instructions, the controller will judge if the user has been provided with enough information to generate an executable plan. When information is sufficient, the controller will call the executor module with an executable task as input (same format as the self-generated one in exploration). The executor module will first analyze the given task referring to the whole skills library, and then generate a high-level plan with controlling code. Otherwise, the observer module will be evoked to ground the user specification with environment information. The input of the observer module is a query string generated by and the observer will try to answer it based on VLM. This process will continue until the controller believes the user requirement is satisfied or can

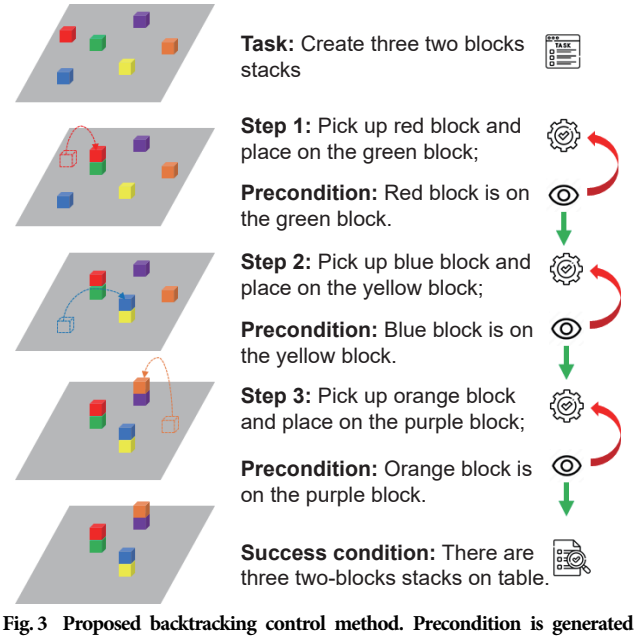


**Fig. 3 Proposed backtracking control method. Precondition is generated for each step which is used to verify if the step executes successfully by VLM.**

not be realized.

When a robot executes its generated plan, there exists the possibility that some steps are not successfully implemented due to errors and online disturbances. We propose backtracking control in the executor module, which automatically generates a precondition for each step, verifies the result of each step, and backtracks to the appropriate step. As shown in Fig. 3, LLM will generate a precondition for each step by considering its context such as the target task and plans for completing this task. We organize the preconditions as object-description text, characterizing the relation between objects or the state of objects. For example, "plate on the microwave" describes the relationship between plate and microwave, and "top drawer is open" describes the state of the top drawer. The verification process utilizes VLM, which is the same as vision-based task verification introduced in Section 2.1.3. If the precondition is not satisfied, the robot will retry by backtracking, that is going back to the nearest step whose precondition meets the current state. Then robot will start from that step and retry the task. We find that the successful rate and robustness of completing a task are significantly improved by absorbing such a close-loop method.

## 3 Experiment

The experiments are designed to validate the following questions:

- Is the proposed framework capable of efficiently guiding a robot to proactively engage with its surroundings and continually develop new skills, thereby enhancing the exploration process?
- Whether the proposed framework has a certain degree of generalization ability? This means that it can be applied to different scenarios and collect useful skills. These skills should be further used to solve related tasks after exploration.
- Can we deploy a robot system in the real world that utilizes skills learned from GExp to accomplish a task specified by the user?

### 3.1 "BLOCKS WORLD"

For the first question, we build the "BLOCKS WORLD" table-top simulation scenario based on RAVENS[37], where six blocks with different colors are placed on the table and a UR5 robot arm is used to interact with the environment. We use "BLOCKS WORLD" for the experiment An ablation experiment is conducted during the process of self-exploration. We compare the proposed method GExp with a framework without collecting skills and a framework without self-verification.

Six different tasks are generated by GExp in "BLOCKS WORLD": pick-and-place single block, one two-block stack, one three-block stack, three two-block stacks, a pyramid with three blocks, and a complex pyramid with six blocks. The details of generated tasks can be found in Appendix B.1. After accomplishing one task, GExp will generate a related skill for further use (see Appendix B.2).

We report the results in Table 1. As the beginning task, moving one block could not be realized successfully because the robot moved the block before lifting it. The self-verification module could help to correct the plan by regenerating it. The following three tasks (blocks stack) show the significant difference between planning with and without using self-generated skills. The last two tasks (pyramid building) go beyond the planning and reasoning ability of LLM with predefined skills. However, GExp enables the robot to solve these complex, long-horizon planning challenges. The average successful rate shows that GExp achieves 0.51 and 0.58 improvement compared with removing skills learning and self-verification modules respectively.

### 3.2 Everyday robotics tasks

To answer the second question, we conduct a series of experiments in RLBench[38] to evaluate the generalization ability of GExp. As shown in Fig. 4, nine different scenes are chosen,

**Table 1 Ablation evaluation results of GExp in "BLOCKS WORLD". The value in the table represents the successful rate of 10 runs. Results show that skills learning and self-verification can facilitate the process of exploration.**

| Task | Successful rate | | |
|---|---|---|---|
| | Without learn skills | Without self-verification | GExp |
| Move one block | 0.8 | 0.4 | 0.8 |
| One two-blocks stack | 0.6 | 0.9 | 1.0 |
| Three two-blocks stack | 0.4 | 1.0 | 1.0 |
| One three-blocks stack | 0.1 | 0.7 | 0.8 |
| Three blocks pyramid | 0.0 | 0.5 | 0.7 |
| Six blocks pyramid | 0.0 | 0.4 | 0.7 |
| Average | 0.32 | 0.25 | 0.83 |

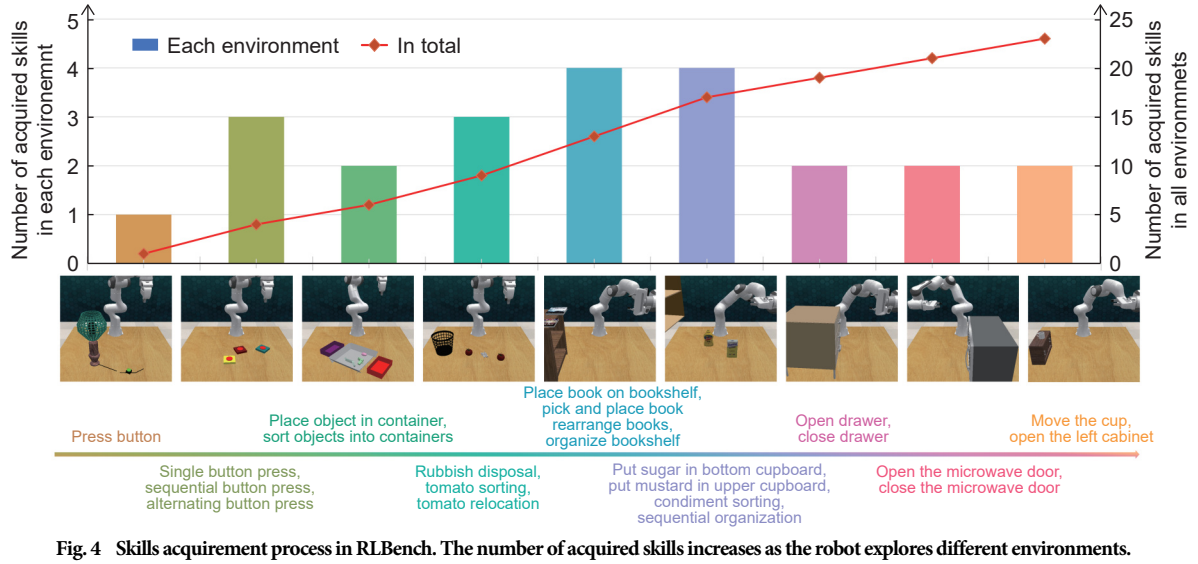


**Fig. 4 Skills acquirement process in RLBench. The number of acquired skills increases as the robot explores different environments.**

including turning on a lamp, pressing buttons, moving object to containers, clearing rubbish, arranging bookshelf, arranging cupboard, manipulating drawer, manipulating microwave, and placing cup to cabinet. The detailed information of different scenes is listed in Appendix C.1.

We first let robots interact with the environments and acquire skills through self-exploration. We arrange 9 environments above according to manipulation complexity and record the number of acquired skills in each environment and the total number of acquired skills across all environments. As is demonstrated in Fig. 4, after the exploration phase, the robot manages to acquire 1 to 4 skills in each environment, with a total of 23 skills across all environments. This result shows that GExp can drive robots to acquire skills efficiently by exploration. Furthermore, we find that there is a relation between the number of acquired skills and the number of objects in the environment, partially because foundation models tend to generate sequential tasks that involve multiple objects.

To evaluate the quality of collected skills in GExp, we set up 2 complex tasks: desktop organization and cup acquisition. Desktop organization involves many objects and needs relatively long-horizon planning. Cup acquisition only involves two objects but the manipulation steps are highly correlated, which means that the success of subsequent actions depends on the success of the current action. More details about the environments can be found in Appendix C.1. We compare GExp with a baseline that is the same as GExp but without backtracking control to show the importance of closed-loop control. We employ the acquired skills to complete these 2 tasks and evaluate the success rate. Moreover, the two tasks can be divided into several sub-tasks, and we also report the success rate of the sub-tasks.

As is shown in Table 2, GExp outperforms baseline in 2 tasks and almost all sub-tasks, especially in difficult sub-tasks such as clear rubbish and close drawer. During the experiment, we found that backtracking control could detect failure and retry the task, which significantly improves the successful rate of completing the whole task. This is particularly beneficial to cup acquisition since this task is highly correlated. We show some backtracking control cases in Appendix C.2. There are also some failures. For example, in cup acquisition, the drawer is regarded as a table sometimes, causing failures in self-verification. It explains why the successful rate of pick & place cup is only 0.5 when using backtracking control. With the development of foundation models, we expect this problem to be mitigated. We include more details in Appendix C.2.

### 3.3 Real-world deployment

For the last question, we set up a real-world tabletop environment letting robot letting robots use a self-learned skills library to accomplish user instruction. As shown in Fig. 5, a UR5 robot arm with 6-DoF equipped with a suction gripper is placed on the table, which can be controlled by using skills learned in Section 3.2. We use a RealSence D415 for perception. The captured RGB image can be used for scene understanding (realized by GPT-4V) and object position acquisition (realized by VLMs including ViLD[39] and Segment Anything[30]).

We sim-to-real two evaluation tasks introduced in Section 3.2: desktop organization and cup acquisition. Different from directly providing a specific task in a simulation environment, a free-form language instruction is given which needs to be transferred into tasks using the workflow introduced in Section 2.3. To further evaluate the generalization ability of GExp, we use objects that are different from those learned in the exploration. For instance, the cup is replaced by a tea caddy, and the objects in the clean table task are a remote controller, plastic bottle, paper, and toy box. Based on the results, we find the proposed framework can successfully transfer user instruction to feasible tasks. The skills can be chosen correctly from the self-learned library and realized the task successfully. The commonsense knowledge of GPT-4V also empowers the self-learned skills generalization ability, where they can be used to manipulate objects not seen before. The experiment log can be found in Appendices D1 and D2. The videos of real-world experiments can be found on our project website: GExp.com.

**Table 2 Open-loop vs. backtracking. We evaluate the performance of robot real-solution on two specially designed tasks. Results show that the proposed backtracking method can significantly improve the successful rate of execution.**

| Task | Successful rate | |
|---|---|---|
| | Open-loop | Backtracking |
| Desktop organization | 0.55 | 0.85 |
| Clear rubbish | 0.60 | 0.90 |
| Place tomato 1 | 0.85 | 0.95 |
| Place tomato 2 | 0.95 | 1.00 |
| Place chocolate jello | 0.90 | 0.90 |
| Place strawberry jello | 1.00 | 1.00 |
| Cup acquisition | 0.20 | 0.45 |
| Open drawer | 0.65 | 0.90 |
| Pick&Place cup | 0.25 | 0.50 |
| Close drawer | 0.60 | 0.85 |


**Fig. 5 Real-world deployment: cup acquisition and desktop organization.**

## 4 Conclusion, Limitation, and Future Work

In this paper, we propose GExp, a novel framework for robot self-exploration utilizing the power of foundation models, which enables robots to explore and learn autonomously without human intervention. To achieve this goal, we devise modules including self-exploration, knowledge-base-building, and close-loop feedback based on foundation models. To validate the effectiveness of the framework, we designed a series of simulation experiments and deployed the framework on a real robot platform. Our results show that GExp allows robots to explore autonomously and learn from exploration by generating useful skills. Based on this framework robots can learn autonomously without relying on human guidance, which is a milestone in the development of robotic intelligence

The main limitation of our method is reliance on the power of foundation models. We find that there still exists a gap between the current ability of foundation models and the requirement of designing a robust and stable system based on them for robotics tasks. GExp is one attempt to implement foundation models enhancing autonomous robots. In the future, we will try to combine learning-based methods into this framework for low-level robot control. We believe the further development of foundation models could further improve the performance of GExp, allowing it to be applied to a wider range of robotics tasks.

## Appendix

### A Module in GExp

#### A.1 How GExp understand the environment?

**Prompt of scene understanding.**

You will be given a picture of table-top scene observation of robot system. You need to give a description about the picture. The description should contain following information:

* What objects are on the table. Please describe those objects.

* You do not need to describe the robot arm, please focus on other objects you observed.

You should output your answer in a json format. The output format should be:

```
```json
{
"Description": "Describe what you have observed on the table.",
"Objects on table":[
    {
        "Name": "The name of object",
        "Color": "object's color",
    },
    ]
}
```
```

**Output of VLM based on observation in Fig. A1.**

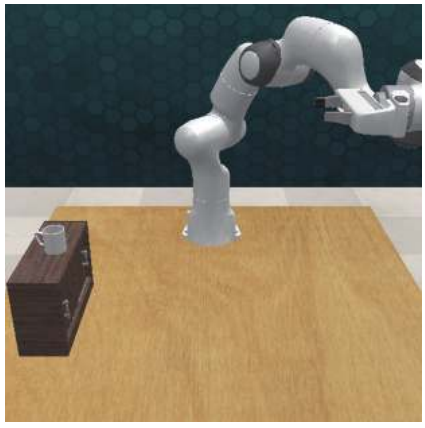

Fig. A1 Initial observation of cabinet environment.

```
```json
{
    "Description": "On the table, there is a wooden brown drawer cabinet situated on the left side. Visual center of the table is almost empty except the robot system which is not the point of interest. Directly on the top side of the drawer, there is a white cup with a handle facing towards the drawer.",
    "Objects on table": [
        {
            "Name": "Drawer cabinet",
            "Color": "Brown"
        },
        {
            "Name": "Cup",
            "Color": "White"
        }
    ]
}
```
```

#### A.2 How GExp generate tasks based on observation?

**Prompt of tasks generation module**

You are a helpful AI assistant. You are helping to create ten different tabletop manipulation tasks according to the provided information. The generated tasks should in the order from easy to hard. The first task should be the easiest one which requires a small number of steps to realize it. The design of the latter task needs to be based on the former. The design purpose of tasks gradually changed from testing basic manipulation ability to long-range planning ability.

The output format should be in the form of json:

```
```json
{
    "Task Name": "The name of generated task",
    "Objects": ["A list of involved objects"],
    "Task Description": "Necessary description of the task"
}
```
```

Rules:

* You must write the full name of object, not just its color.

* Don't use "all objects", you should write them one by one.

* Task should involved robot manipulation not just observation.

* Do not create task with any assume value of distance or position, you can use relative value to the boundary.

* You should output ```json and ``` showing the output is json block.

#### A.3 How GExp realize planning and controlling the robot?

**Prompt of planning module.**

You are a sophisticated AI assistant tasked with creating a multi-step plan for a robot manipulation task. Your role is to design this plan methodically, ensuring accuracy and precision. To successfully execute this task, please adhere to the following guidelines and rules:

* Library Restrictions: The only permitted library is "numpy".

Please import it as "np" if necessary. Avoid using any other libraries.

* Use Existing Functions: Do not create new functions. Only utilize the functions that are provided.

* Variable Constraints: Avoid using variables that require user specification or complex calculations. Instead, derive exact values based on the given variables.

* Position and Size Considerations: Do not make assumptions about positions or sizes.

* Adherence to BOUNDS: Ensure that all positions and movements fall within the robot arm's working BOUNDS. When establishing new positions, reference the BOUNDS information for accuracy.

* Completeness of Plan: You should output the whole plan without using the expression like "repeat steps 2–5 to...".

* Efficient Function Usage: Some high-level functions are provided to help you finish the task. You should try to use them to optimize the plan.

Format for the Plan:

Your plan should be structured in a series of JSON blocks, each representing a step in the process. The format for each block is as follows:

```json
{
    "Name": "Provide a name for this step.",
    "Explanation": "Include a necessary explanation about this step.",
    "Code": "Write the Python code to execute this step. You should write the code in a single line using '\\n' for changing line."
}
```

*Note: The number of JSON blocks should correspond to the number of steps in your plan. You should output ```json and ``` showing the output is json block.

Please remember that you can only use following defined functions:

### A.4 How GExp verify the precondition or task success or not with VLM?

**Prompt of verification module.**

You are a helpful AI assistant. Your job is to determine if a condition is satisfied or not. You will be given a description of the condition and an RGB image capturing the current situation. Please determine it in a step-by-step way to be sure that we have the right answer.

To help you understand the scene, I will give you a description of the initial scene setup and a corresponding image. So you will get two images, the first is the initial scene image, and the second is the current scene image.

```json
{
    "condition": "The box is grasped by the robot arm",
    "initial description": f"{VLM_initial_observation}",
}
```

**Output of VLM based on the observation in Fig. A2.**

To determine if the condition "The box is grasped by the robot arm" is satisfied, let's consider the two images provided:

1. Initial scene description image: This image displays the initial setup with the cupboard, the robot arm, the table, the box, and the yellow bottle. The box is on the table, to the left of the yellow bottle, and not being grasped by the robot arm, which is positioned to the right side of the box, with the gripper open and not in contact with the box.

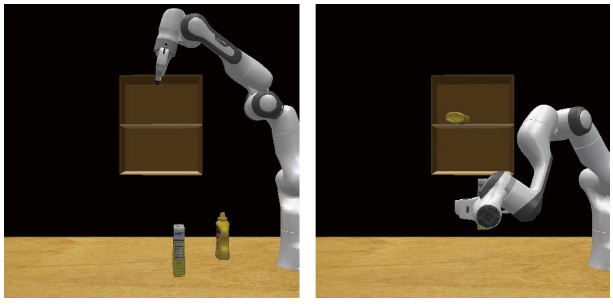

**Fig. A2 Initial observation and current observation of cabinet environment. The verification condition is whether the robot arm has grasped the milk box.**

2. Current scene description image: This second image shows a different state where the robot arm has moved from its initial position, and the end-effector (gripper) of the robot arm appears to be surrounding the box, indicating that the arm has attempted to grab it. The yellow bottle is within the box, suggesting some movement of objects has occurred. It is not possible to confirm a firm grasp visually without additional cues like the arm's grip pressure or the box's weight distribution. However, the change in position coupled with the gripper's placement around the box strongly suggests that the robot arm is now holding the box.

Given this analysis, I can conclude that the condition "The box is grasped by the robot arm" is met.

### A.5 How GExp generate skill from a successful experience?

**Prompt of skill generation.**

You are a helpful AI assistant expertise in programming robot controlling code. You will be given with a Python script which is used to solve a robot manipulation problem. Your task is creating one Python function that helps people in future solve similar task in an easier way. Your function will be send into function library for further use.

Following rules you should follow when creating new function:

* Library Restrictions: The only permitted library is "numpy". Please import it as "np" if necessary. Avoid using any other libraries.

* Use Existing Functions: When creating new function, you should use existing functions in library.

* Function should be general: The generated new function should be general. Do not generate a function to realize the given specific task.

Format of your response:

You should output a json block as your output, which follows the following format:

```json
{
    "Function name": "Name_of_your_function(), for example lift_object().",
    "Description": "Describe the function your created.",
    "Input": "What is the input of your function.",
    "Output": "What is the output of your function.",
    "Code": "The code of generated function.",
    "Example": "You should rewrite the code using new function.",
    "Related functions": "what functions are used in this function."
}
```

```
Existing functions in function library:
f"{existing_skills_library}"
```

## B "BLOCKS WORLD" Experiment

### B.1 Tasks generated in "BLOCKS WORLD"

**Output of tasks generate module.**

```
{
    {
        "Task Name": "Pick and Place the Purple Block",
        "Objects": [
            "purple block"
        ],
        "Task Description": "The robot arm should identify the purple block, pick it up, and place it at the right boundary of the table."
    },
    {
        "Task Name": "Create a Two-Block Stack",
        "Objects": [
            "purple block",
            "blue block"
        ],
        "Task Description": "The robot needs to stack the purple block on top of the blue block."
    },
    {
        "Task Name": "Create a Three-Block Stack",
        "Objects": [
            "purple block",
            "blue block",
            "green block"
        ],
        "Task Description": "The robot arm should pick up the green block and place it on top of the stack of the purple and blue blocks."
    },
    {
        "Task Name": "Color Match and Stack",
        "Objects": [
            "purple block",
            "blue block",
            "green block",
            "yellow block",
            "orange block",
            "red block"
        ],
        "Task Description": "The task is to create three stacks of two blocks each, where each stack contains blocks of complementary colors. For example, one stack could have the purple and yellow blocks, the second could have the blue and orange blocks, and the third could have the green and red blocks."
    },
    {
        "Task Name": "Block Pyramid Stacking",
        "Objects": [
            "yellow block",
            "orange block",
            "red block"
        ],
        "Task Description": "The robot needs to stack the blocks into a pyramid shape in the center of the table. The bottom layer would consist of the yellow and orange blocks side by side, and the red block should be placed on top."
    },
    {
        "Task Name": "Complex Pyramid Construction",
        "Objects": [
            "purple block",
            "blue block",
            "green block",
            "yellow block",
            "orange block"
        ],
        "Task Description": "The robot needs to stack all the blocks into a pyramid with three levels in the center of table. The bottom level should consist of the purple, blue, and green blocks. The middle level should have the yellow and orange blocks. The red block should be on top. The robot needs to plan the sequence and positions of its movements carefully to build the pyramid without knocking any blocks over."
    }
}
```

### B.2 Skills generated in "BLOCKS WORLD"

```
{
  "pick_and_place_object(object_name, destination)": {
    "Type": "function",
    "Description": "This function is used to pick and place a specific object from its current location to a specified destination. The robot identifies the object, picks it up, and places it at the destination.",
    "Input": "('object_name': str, 'destination': list). 'object_name' is the name of the object to be moved, 'destination' is a list of 3 elements representing the coordinates where the object will be placed.",
    "Output": "None. The function only performs actions, it does not return any value.",
    "Related functions": "get_obj_position(), get_obj_dimensions(), movep(), close_gripper(), open_gripper(), go_home()",
    "Code": "def pick_and_place_object(object_name, destination): \\n object_position = get_obj_position(object_name) \\n object_dimensions = get_obj_dimensions(object_name)\\n BOUNDS=[[0.25,0.75],[-0.5,0.5],[0.0,0.28]]\\n movep((object_position[0], object_position[1], object_position[2] + object_dimensions[2])) \\n movep((object_position[0], object_position[1], object_position[2])) \\n close_gripper() \\n movep((object_position[0],object_position[1],BOUNDS[2][1])) \\n movep((destination[0], destination[1], BOUNDS[2][1])) \\n movep((destination[0], destination[1], BOUNDS[2][0] + destination[2])) \\n open_gripper() \\n go_home() \\n "
  },
  "stack_blocks(block1, block2)": {
    "Type": "function",
    "Description": "This function is designed to stack a specific block on top of another block. It uses robot functions such as 'go_home()', 'pick_and_place_object()', 'get_obj_position()' and 'get_obj_dimensions()' to achieve this task. This function will help automate the process of block stacking in future robot manipulation tasks.",
    "Input": "Two strings, block1 and block2, which represent the names of the blocks to be stacked.",
    "Output": "None. The function performs the action of moving the robot to stack the blocks but doesn't return
```

anything.",
"Related functions": "go_home(), pick_and_place_object(), get_obj_position(), get_obj_dimensions()",
"Example": "stack_blocks('purple block', 'blue block')",
"Code": "def stack_blocks(block1, block2):\\n go_home()\\n pick_and_place_object(block1, list(np.array(get_obj_position(block2)) + np.array([0, 0, get_obj_dimensions(block2)[2]])))\\n go_home()"
},
"create_three_block_stack()": {
"Type": "function",
"Description": "This function is used to create a three-block stack. It first stacks the first two blocks, then calculates the position to place the third block, moves the third block to the stack, and finally, the robot returns to the home position.",
"Input": "Three parameters: first two blocks for initial stacking and the third block to be placed on top of the stack.",
"Output": "The function does not return any output. Its purpose is to perform the action of creating a three-block stack.",
"Related functions": "stack_blocks(), get_obj_position(), get_obj_dimensions(), pick_and_place_object(), go_home()",
"Example": "Example usage: create_three_block_stack ('purple block', 'blue block', 'green block')",
"Code": "def create_three_block_stack(first_block, second_block, third_block): \\n # Step 1: Stack first and second Blocks\\n stack_blocks(first_block,second_block)\\n #Step 2: Get Position and Dimensions of the Stack \\n stack_position = get_obj_position(second_block) \\n stack_dimensions = get_obj_dimensions(second_block) \\n # Step 3: Calculate Position to Place third Block \\n third_block_position = (stack_position[0], stack_position[1], stack_position[2] + stack_dimensions[2]/2 + get_obj_dimensions(third_block)[2]/2) \\n # Step 4: Move third Block to the Stack \\n pick_and_place_object(third_block, third_block_position) \\n # Step 5: Return to Home Position \\n go_home()"
},
"build_pyramid(blocks_order)": {
"Type": "function",
"Description": "This function is designed to build a pyramid shape with the blocks. It takes an array of block names in the order from bottom to top. The first two blocks will be placed side by side at the bottom. The next block will be placed on top of the first two blocks.",
"Input": "An array of block names, from bottom to top. For example, ['yellow block', 'orange block', 'red block'] means yellow and orange blocks will be placed at the bottom and red block on top.",
"Output": "No output. The function commands the robot to build the pyramid according to the block order.",
"Related functions": "Functions used in this function are: pick_and_place_object(),get_obj_position(),get_obj_dimensions(), go_home().",
"Example": "Example of usage: build_pyramid(['yellow block', 'orange block', 'red block'])",
"Code": "def build_pyramid(blocks_order): \\n # Step 1: Move first block to the right boundary \\n pick_and_place_object(blocks_order[0], [BOUNDS[0][1], BOUNDS[1][0], BOUNDS[2][0]]) \\n # Step 2: Move second block next to the first block \\n first_block_position = get_obj_position(blocks_order[0]) \\n first_block_dimensions = get_obj_dimensions (blocks_order[0])\\n second_destination=[first_block_position[0]+ first_block_dimensions[0], first_block_position[1], first_block_position[2]] \\n pick_and_place_object(blocks_order[1], second_destination) \\n # Step 3: Stack third block on top of the first and second blocks \\n second_block_position = get_obj_position(blocks_order[1]) \\n third_destination = [(first_block_position[0] + second_block_position[0]) / 2, first_block_position[1], first_block_position[2] + first_block_dimensions[2]] \\n pick_and_place_object(blocks_order[2], third_destination) \\n # Step 4: Return to home position \\n go_home()"
}
}

## C RLBench Experiment

### C.1 RLBench-based environments

To test the capability of GExp in daily tasks, we first utilize nine environments in RLBench, which are shown in Fig. A3.

Turning on a Lamp: There are a lamp and a button on the table. Robot can turn the lamp on by pressing the button.

Pressing Buttons: There are 3 buttons on the table. Robot can press them sequentially. This process can be seen in daily tasks, such as microwaving food. This environment is more complex than Turning on a Lamp.

Moving Object to Containers: One large container and two small containers are on the table. Several objects are in the larger container. Robot can explore and acquire some basic picking or placing skills in this environment.

Clearing Rubbish: One rubbish bin, two tomatoes and one paper rubbish are on the table. Robot is hoped to clear the rubbish by picking and placing it into the bin.

Arranging Bookshelf: A bookshelf is placed on the table. Two books are on the top of the bookshelf. Robot can try to place them

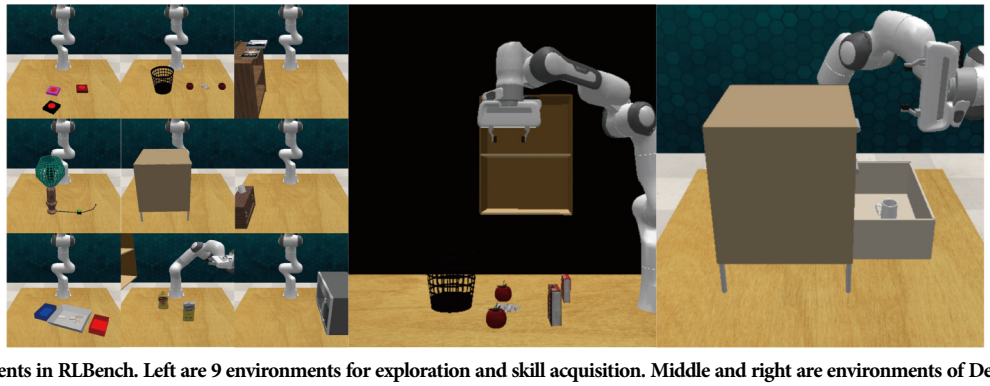

**Fig. A3 Environments in RLBench. Left are 9 environments for exploration and skill acquisition. Middle and right are environments of Desktop Organization and Cup Acquisition respectively.**

in the bookshelf.

Arranging Cupboard: A sugar box and a milk bottle are on the table. A cupboard is near the table. The sugar box and the milk bottle can be transferred into the cupboard by robot.

Manipulating Drawer: A three-layer drawer is put on the table. Different from above environments, this environment involves articulated object.

Manipulating Microwave: A microwave is placed on the table. Robot can try to open and close it.

Placing Cup to Cabinet: A cabinet and a cup are on the table. The cabinet has two doors, which can be opened by sliding in left or right direction. This task combines articulated objects with non-articulated objects.

To further prove the importance of acquired skills and backtracking control, we also build two complex environments and tasks. Environments are shown in Fig. A3.

Desktop Organization: We combine Clear Rubbish with Arranging Cupboard. A paper rubbish, a rubbish bin, two tomatoes, a chocolate jello box and a strawberry jello box are placed on the table. A cupboard is positioned nearby. In this task, we require robot to clean rubbish and organize other objects to proper positions. This task mainly tests the long-horizon planning and skill usage of GExp.

Cup Acquisition: There are a cup and drawer in this scene. The cup is placed in the bottom drawer and is not observable. We require robot to find the cup by opening drawer and place the cup on the table. In the end, robot should close the drawer. This task is especially difficult since the steps are highly correlated, as mentioned in 3.2.

## C.2 Specific cases in RLBench-based environments

In this section, we provide some cases on backtracking-based close-loop control and failures.

As seen in Fig. A4, robot fails to place the rubbish in the rubbish bin due to unstable grasping. However, with VLM as a verifier and the precondition generated by LLM, robot can detect the failure and retry to clear the rubbish. Some similar cases can be also observed in Cup Acquisition task. For example, as is shown in Fig. A5, robot does not open the drawer fully, close the drawer, or loosens the cup occasionally, causing the rest of task unfinished. Backtracking-based close-loop control endows robot to reflect and adjust according to current state.

Nevertheless, GExp also fails in some cases, which are illustrated in Fig. A6. In Desktop Organization, one of boxes is knocked over due to unstable grasping. This results unreachable poses. Therefore, in the subsequent steps, robot fails to pick up the box. In Cup Acquisition Organization, failures occurs more frequently, particularly in opening drawer and picking up cup. When robot executes these two steps, there is dilemma that the drawer is partially open but robot can not pick it, as is shown in

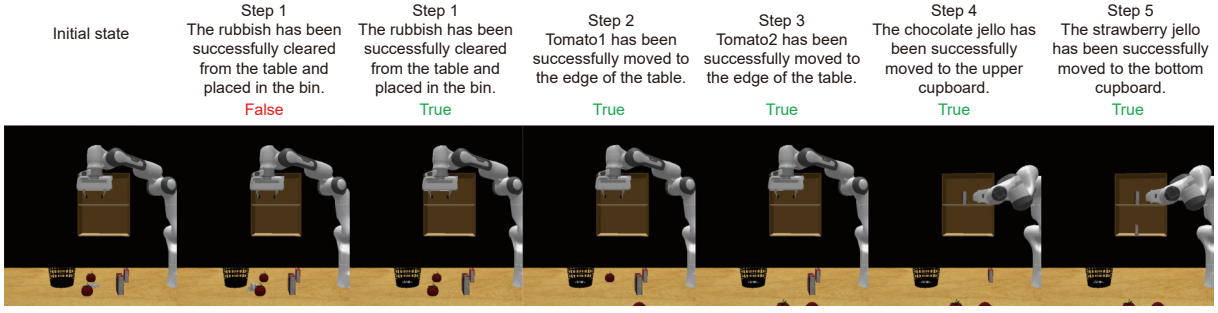


**Fig. A4 Close-loop case in Desktop Organization task. The robot manages to detect failure and retry clearing rubbish.**

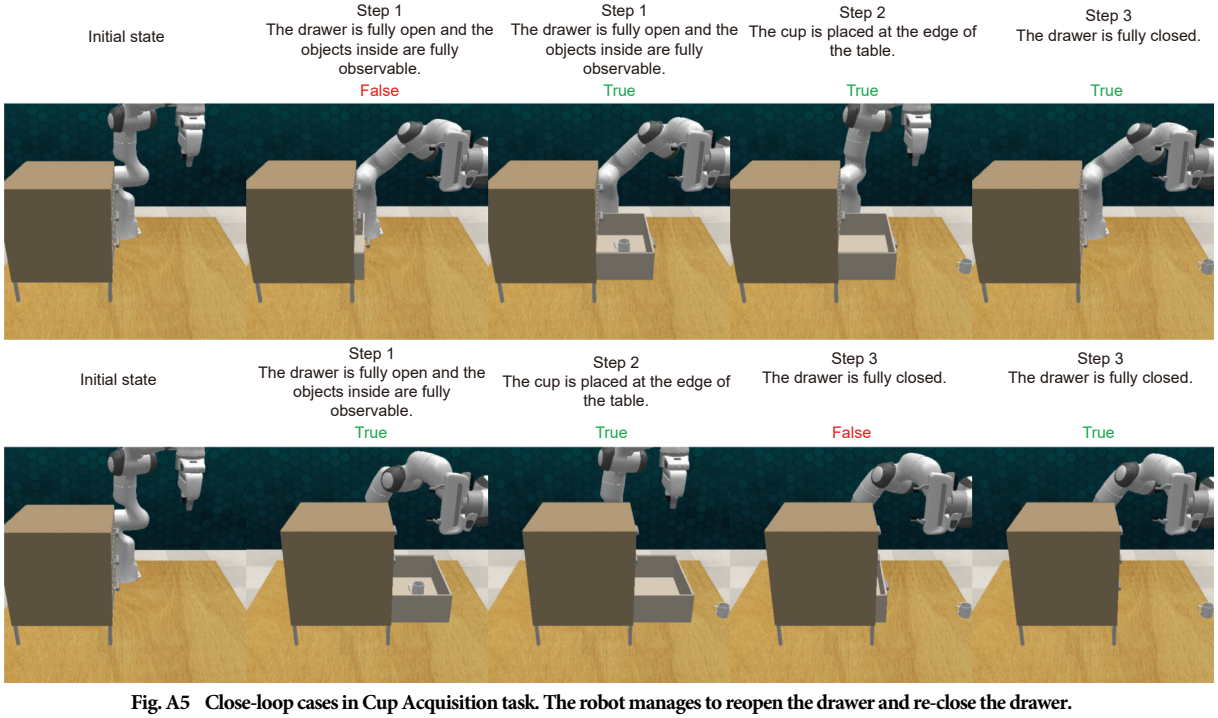


**Fig. A5 Close-loop cases in Cup Acquisition task. The robot manages to reopen the drawer and re-close the drawer.**

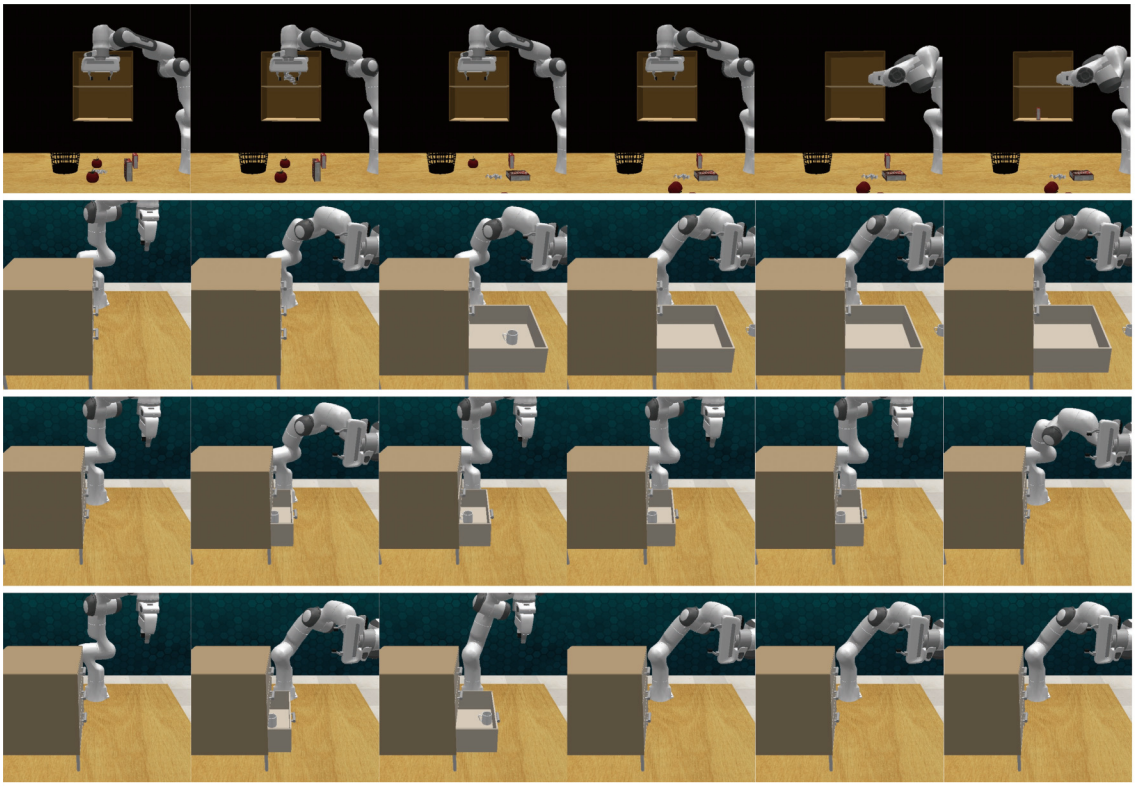

**Fig. A6 Failure cases. The falling rubbish knocks down a box, resulting ungraspable pose for the robot. Second row: the drawer is opened so large that the robot can not fetch the handle (the handle is not in the reachable space of the robot arm). Third row: The robot thinks the drawer is open but it is not opened fully so the robot can not pick the cup up. Fourth row: The robot regards the drawer as a table, not realizing the failure of placing the cup.**

the third row of Fig. A6. Moreover, VLM malfunctions sporadically, regarding the drawer as a table. And randomness in manipulation will also cause unrecoverable states.

## D Real-World Experiment

### D.1 Finding the cup

**Conversation log of finding cup task.**

User:
I can't find my tea caddy.
Controller:
{
"Thought": "The user can't find their tea caddy. The first step is to observe the environment to locate the missing item.",
"Action": "observe()",
"Action input": {"query": "Where is the tea caddy?"}
}
Observer:
{
"answer": "The tea caddy is not visible in the picture.",
"observed_objs": ["drawer"]
}
Controller:
{
"Thought": "The tea caddy isn't currently visible. However, a drawer was observed in the environment. It's possible that the tea caddy might be inside the drawer.",
"Action": "observe()",
"Action input": {"query": "What is inside the drawer?"}
}
Observer:
{
"answer": "The image does not show the inside of the drawer, so it is impossible to determine what is inside it.",
"observed_objs": ["drawer"]
}
Controller:
{
"Thought": "Since I can't see inside the drawer, the next logical step would be to open it and check if the tea caddy is inside.",
"Action": "execute_task()",
"Action input": {"Task name": "Open the drawer", "Objects": ["drawer"], "Task description": "The robot arm needs to open the drawer to check if the tea caddy is inside."}
}
Executor:
"open_drawer('drawer')"
Observer:
{
"Thought": "The drawer has been opened successfully. Now I need to observe the contents of the drawer to see if the tea caddy is inside.",
"Action": "observe()",
"Action input": {"query": "What is inside the drawer now?"}
}
Observer:
{
"answer": "The tea caddy is in the drawer.",
"observed_objs": ["drawer", "tea caddy"]
}
Controller:
{

"Thought": "The tea caddy has been observed inside the drawer. Now I should inform the user about the location of the tea caddy.",
"Action": "finish()",
"Action input": {"message": "I found your tea caddy. It is in the drawer."}
}
user:
Now the drawer is open, you don't need to open again. Move tea caddy to edge of table and close drawer.
Controller:
{
"Thought": "The user wants me to move the tea caddy to the edge of the table and then close the drawer. I will first move the tea caddy to the specified location.",
"Action": "execute_task()",
"Action input": {"Task name": "Move the tea caddy", "Objects": ["tea caddy"], "Task description": "The robot arm should move the tea caddy to the edge of the table."}
}
Executor:
"move_object_to_edge('tea caddy', BOUNDS)"
"close_drawer('drawer')"
Controller:
{
"Thought": "The tea caddy has been moved to the edge of the table successfully. Now, I need to close the drawer as per the user's request.",
"Action": "execute_task()",
"Action input": {"Task name": "Close the drawer", "Objects": ["drawer"], "Task description": "The robot arm should close the drawer."}
}
Executor:
"close_drawer('drawer')"
Controller:
{
"Thought": "I've moved tea caddy to edge of table and close drawer. The task should be finished.",
"Action": "finish()",
"Action input": {"message": "I've moved tea caddy to edge of table and close drawer."}
}

### D.2 Cleaning the table

User:
Please clean the table: put rubbish into rubbish bin and put useful objects on the shelf.
Controller:
{
"Thought": "First, I need to analyze the environment to observe the objects on the table, and identify which ones are rubbish and which ones are not.",
"Action": "observe()",
"Action input": {"query": "What objects are on the table?"
}
}
Observer:
{
"answer": "The objects on the table include a green basket, a remote control, a water bottle, a white paper, and a yellow box.",
"observed_objs": ["basket", "remote control", "water bottle", "paper", "box"]
}
Controller:
{
"Thought": "Next, I need to identify which objects on the table are rubbish. Usually, papers could be considered as rubbish. For the other objects, I need to confirm.",
"Action": "observe()",
"Action input": {"query": "Which objects on the table are rubbish?"}
}
Observer:
{
"answer": "The plastic bottle is rubbish.",
"observed_objs": ["plastic bottle", "paper"]
}
Controller:
{
"Thought": "Now that I know which objects are rubbish (the plastic bottle and the paper), I need to find the rubbish bin to put them in.",
"Action": "observe()",
"Action input": {"query": "Where is the rubbish bin?"}
}
Observer:
{
"answer": "The rubbish bin is in the middle of the table, slightly towards the left side.",
"observed_objs": ["water bottle", "remote", "paper", "box"]
}
Controller:
{
"Thought": "Now I know where the rubbish bin is. I will start by moving the rubbish into the bin. First, I will move the plastic bottle, followed by paper.",
"Action": "execute_task()",
"Action input": {
"Task name": "Move rubbish into bin",
"Objects": ["plastic bottle", "paper", "rubbish bin"],
"Task description": "Pick up the plastic bottle and paper, and put them into the rubbish bin"
}
}
Executor:
# Move Bottle to Bin
pick_and_place_object('bottle', get_obj_position('bottle'), get_obj_position('bin'))
# Move Paper to Bin
pick_and_place_object('paper', get_obj_position('paper'), get_obj_position('bin'))}
Controller:
{
"Thought": "I've put the rubbish into the bin successfully. Now, I need to move the remaining objects to the shelf. But first, I need to locate the shelf.",
"Action": "observe()",
"Action input": {"query": "Where is the shelf?"}
}
Observer:
{
"answer": "The shelf is on the right side of the table.",
"observed_objs": ["cup", "remote", "water bottle", "paper", "box"]

}
Controller:
{
"Thought": "Now I know where the shelf is. I will start by moving the remaining objects to the shelf. First, I will move the remote, followed by the box.",
"Action": "execute_task()",
"Action input": {
"Task name": "Move objects to shelf",
"Objects": ["remote control", "box", "shelf"],
"Task description": "Pick up the remote control and the box, and put them onto the shelf"
}
}
Executor:
# Move the remote control to shelf_1
pick_and_place_object('remote control', get_obj_position ('remote control'), get_obj_position('shelf_1'))
# Move the box to shelf_2
pick_and_place_object('box', get_obj_position('box'), get_obj_ position('shelf_2'))
COntroller:
{
"Thought": "I've moved all the objects to the shelf and put the rubbish into the bin. The table should now be clean.",
"Action": "finish()",
"Action input": {"message": "I have cleaned the table by putting rubbish into the rubbish bin and moving the objects onto the shelf."}
}

## Acknowledgment

This work was supported by Shenzhen Key Laboratory of Ubiquitous Data Enabling (No. ZDSYS20220527171406015), Guangdong Innovative and Entrepreneurial Research Team Program (No. 2021ZT09L197), Shenzhen Science and Technology Program (No. JCYJ20220530143013030), National Natural Science Foundation of China (No. 62104125), Tsinghua Shenzhen International Graduate School-Shenzhen Pengrui Young Faculty Program of Shenzhen Pengrui Foundation (No. SZPR2023005), and Shenzhen Higher Education Stable Support Program (No. WDZC20231129093657002).

## Article History

Received: 24 January 2024; Revised: 23 February 2024; Accepted: 22 March 2024

## References

[1] G. Hacques, J. Komar, M. Dicks, and L. Seifert, Exploring to learn and learning to explore, *Psychol. Res.*, vol. 85, no. 4, pp. 1367–1379, 2021.
[2] S. Ivaldi, S. M. Nguyen, N. Lyubova, A. Droniou, V. Padois, D. Filliat, P. -Y. Oudeyer, and O. Sigaud, Object learning through active exploration, *IEEE Trans. Auton. Mental Dev.*, vol. 6, no. 1, pp. 56–72, 2014.
[3] M. Balsells, M. Torne, Z. Wang, S. Desai, P. Agrawal, and A. Gupta, Autonomous robotic reinforcement learning with asynchronous human feedback, in *Proc. 7th Conf. Robot Learning* (*CoRL 2023*), Atlanta, GA, USA, 2023, pp. 774–799.
[4] M. Wahde, Introduction to Autonomous Robots, https://www.me.chalmers.se/~mwahde/courses/aa/2016/FFR125_LectureNotes.pdf, 2016.
[5] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlavac, T. -K. Kim, and S. Malassiotis, Folding clothes autonomously: A complete pipeline, *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1461–1478, 2016.
[6] C. Bersch, B. Pitzer, and S. Kammel, Bimanual robotic cloth manipulation for laundry folding, in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, San Francisco, CA, USA, 2011, pp. 1413–1419.
[7] K. Mo, Y. Deng, C. Xia, and X. Wang, Learning language-conditioned deformable object manipulation with graph dynamics, arXiv preprint arXiv: 2303.01310, 2023.
[8] H. Shi, H. Xu, S. Clarke, Y. Li, and J. Wu, Robocook: Long-horizon elasto-plastic object manipulation with diverse tools, arXiv preprint arXiv: 2306.14447, 2023.
[9] Z. Fu, T. Z. Zhao, and C. Finn, Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, arXiv preprint arXiv: 2401.02117, 2024.
[10] Z. Yan, N. Crombez, J. Buisson, Y. Ruichek, T. Krajnik, and L. Sun, A quantifiable stratification strategy for tidy-up in service robotics, *in Proc. IEEE Int. Conf. Advanced Robotics and Its Social Impacts* (*ARSO*), Tokoname, Japan, 2021, pp. 182–187.
[11] I. Kapelyukh and E. Johns, My house, my rules: Learning tidying preferences with graph neural networks, in *Proc. 5th Conf. Robot Learning* (*CoRL 2021*), London, UK, 2023, pp. 740–749.
[12] S. Li, H. Yu, W. Ding, H. Liu, L. Ye, C. Xia, X. Wang, and X. -P. Zhang, Visual–tactile fusion for transparent object grasping in complex backgrounds, *IEEE Trans. Robot.*, vol. 39, no. 5, pp. 3838–3856, 2023.
[13] Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, H. Liu, and F. Sun, Deep reinforcement learning for robotic pushing and picking in cluttered environment, in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems* (*IROS*), Macau, China, 2019, pp. 619–626.
[14] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods, *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, 2017.
[15] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, et al., GPT-4 technical report, arXiv preprint arXiv: 2303.08774, 2023.
[16] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv: 2307.09288, 2023.
[17] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and F. F. Li, Voxposer: Composable 3D value maps for robotic manipulation with language models, arXiv preprint arXiv: 2307.05973, 2023.
[18] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, Code as policies: Language model programs for embodied control, in *Proc. IEEE Int. Conf. Robotics and Automation* (*ICRA*), London, UK, 2023, pp. 9493–9500.
[19] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances, in *Proc. 6th Conf. Robot Learning* (*CoRL 2022*), Auckland, New Zealand, 2022, pp. 287–318.
[20] W. K. Vong, W. Wang, A. E. Orhan, and B. M. Lake, Grounded language acquisition through the eyes and ears of a single child, *Science*, vol. 383, no. 6682, pp. 504–511, 2024.
[21] X. Zhu, Y. Chen, H. Tian, C. Tao, W. Su, C. Yang, G. Huang, B. Li, L. Lu, X. Wang, et al., Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory, arXiv preprint arXiv: 2305.17144, 2023.
[22] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, Voyager: An open-ended embodied agent with large language models, arXiv preprint arXiv: 2305.16291, 2023.
[23] Y. Hu, F. Lin, T. Zhang, L. Yi, and Y. Gao, Look before you leap: Unveiling the power of GPT-4v in robotic vision-language planning, arXiv preprint arXiv: 2311.17842, 2023.

[24] Z. Yang, L. Li, K. Lin, J. Wang, C. C. Lin, Z. Liu, and L. Wang, The dawn of LMMs: Preliminary explorations GPT-4v (ision), arXiv preprint arXiv: 2309.17421, 2023.

[25] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, ChatGPT for robotics: Design principles and model abilities, *IEEE Access*, vol. 12, pp. 55682–55696, 2024.

[26] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al., Inner monologue: Embodied reasoning through planning with language models, arXiv preprint arXiv: 2207.05608, 2022.

[27] T. Kwon, N. Di Palo, and E. Johns, Language models as zero-shot trajectory generators, arXiv preprint arXiv: 2310.11604, 2023.

[28] M. Xu, P. Huang, W. Yu, S. Liu, X. Zhang, Y. Niu, T. Zhang, F. Xia, J. Tan, and D. Zhao, Creative robot tool use with large language models, arXiv preprint arXiv: 2310.13065, 2023.

[29] Y. J. Ma, W. Liang, G. Wang, D. A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, Eureka: Human-level reward design via coding large language models, arXiv preprint arXiv: 2310.12931, 2023.

[30] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C Berg, W. Y. Lo, et al., Segment anything, arXiv preprint arXiv: 2304.02643, 2023.

[31] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, et al., Simple open-vocabulary object detection with vision transformers, arXiv preprint arXiv: 2205.06230, 2022.

[32] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, Integrated task and motion planning, *Annu. Rev. Control Robot. Auton. Syst.*, vol. 4, pp. 265–293, 2021.

[33] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, SHOP: Simple hierarchical ordered planner, in *Proc. 16th Int. Joint Conf. Artificial Intelligence* (*IJCAI'99*), Stockholm, Sweden, 1999, pp. 968–973.

[34] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh, Translating natural language to planning goals with large language models, arXiv preprint arXiv: 2302.05128, 2023.

[35] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in *Proc. 36th Conf. Neural Information Processing Systems* (*NeurIPS 2022*), New Orleans, LA, USA, 2022, pp. 24824–24837.

[36] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, ReAct: Synergizing reasoning and acting in language models, arXiv preprint arXiv: 2210.03629, 2022.

[37] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, et al., Transporter networks: Rearranging the visual world for robotic manipulation, in *Proc. 4th Conf. Robot Learning* (*CoRL 2020*), virtual, 2020.

[38] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, RLBench: The robot learning benchmark & learning environment, *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 3019–3026, 2020.

[39] X. Gu, T. Y. Lin, W. Kuo, and Y. Cui, Open vocabulary object detection via vision and language knowledge distillation, arXiv preprint arXiv: 2104.13921, 2021.