RoboCodeX: Multimodal Code Generation for Robotic Behavior Synthesis

Yao Mu [1 2] **Junting Chen** [2 3] **Qinglong Zhang** [2] **Shoufa Chen** [1] **Qiaojun Yu** [4] **Chongjian GE** [1] **Runjian Chen** [1 2]
Zhixuan Liang [1] **Mengkang Hu** [1 2] **Chaofan Tao** [1] **Peize Sun** [1] **Haibao Yu** [1] **Chao Yang** [2] **Wenqi Shao** [2]
Wenhai Wang [2 5] **Jifeng Dai** [2 6] **Yu Qiao** [2] **Mingyu Ding** [7] **Ping Luo** [1 2]

Abstract

Robotic behavior synthesis, the problem of understanding multimodal inputs and generating precise physical control for robots, is an important part of Embodied AI. Despite successes in applying multimodal large language models for high-level understanding, it remains challenging to translate these conceptual understandings into detailed robotic actions while achieving generalization across various scenarios. In this paper, we propose a tree-structured multimodal code generation framework for generalized robotic behavior synthesis, termed RoboCodeX. RoboCodeX decomposes high-level human instructions into multiple object-centric manipulation units consisting of physical preferences such as affordance and safety constraints, and applies code generation to introduce generalization ability across various robotics platforms. To further enhance the capability to map conceptual and perceptual understanding into control commands, a specialized multimodal reasoning dataset is collected for pre-training and an iterative refining methodology is introduced for supervised fine-tuning. Extensive experiments demonstrate that RoboCodeX achieves state-of-the-art performance in both simulators and real robots on four different kinds of manipulation tasks and one embodied navigation task. More demos and information can be found in our homepage.

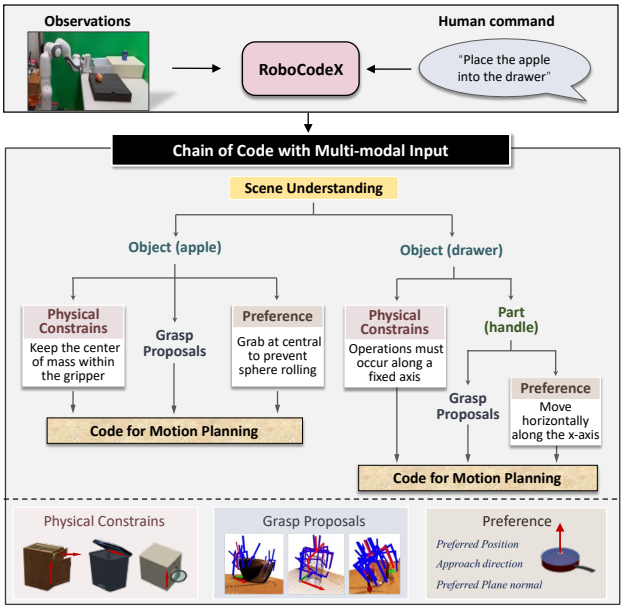

Figure 1. An illustration of our RoboCodeX, a large vision language model with tree-of-thought reasoning capabilities for robotic code generation. It decomposes high-level human instructions into multiple object-focused manipulation subtasks, further expanding them by predicting physical constraints, preferential rankings, and target position proposals. RoboCodeX acts as an interface to translate high-level semantic understanding of the observed environment and instructions into tailored robotic behaviors.

1. Introduction

Embodied AI equips intelligent agents, such as robots, with the capacity for perception, reasoning, and interaction within the 3D physical world and with humans. Yet, a central challenge lies in the generalizability of robotic manipulation frameworks. On one hand, robot policies may struggle to generalize to objects with diverse properties and mechanisms, *e.g.*, objects with different physical constraints and grasping preferences in Figure. 1. On the other hand, adapting a robot learning framework to different robotic platforms encounters difficulties. These challenges stem from the gap between the high-level scene understanding and the low-level manipulation and control policies. How to create a general framework with reasoning capabilities that

[1]Department of Computer Science, The University of Hong Kong, Hong Kong [2]OpenGVLab, Shanghai AI Laboratory [3]ETH Zurich [4]Shanghai Jiao Tong University [5]The Chinese University of Hong Kong [6]Tsinghua University [7]UC Berkely. Correspondence to: Ping Luo <pluo@cs.hku.hk>, Mingyu Ding <myding@berkeley.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

can map the high-level semantic understanding to generic robotics behaviors, thus becomes a primary concern.

Previous methods (Ahn et al., 2022; Huang et al., 2023b; Rana et al., 2023) leverage the reasoning capabilities of Large Language Models (LLMs) to propose step-by-step natural language plans for robot execution. However, lacking environmental information (*e.g.*, status and shape of objects from vision) as contextual grounding, it is difficult to leverage LLMs for decision-making within a given real-world context. Consequently, they rely heavily on low-level skill models (Brohan et al., 2022; 2023) to convert visual observations and language commands into low-level actions. Nonetheless, such skill models still struggle to generalize to novel scenarios or robotic platforms, which requires precise tuning of action sequences tailored to the robot's morphology and actuators.

Specifically, robotic skills and low-level behaviors for different tasks encompass these precise action sequences that fulfill proper contact points, grasping poses, physical constraints, waypoints, and movement paths. Though previous methods (Gao et al., 2023; Mirjalili et al., 2023; Agrawal et al., 2023; Mo et al., 2019; Xu et al., 2022; Eisner et al., 2022; Zhong et al., 2023; Li et al., 2023e) have explored estimating manipulation preferences, such as grasp regions of an object (Mirjalili et al., 2023) and contact points (Li et al., 2023e), their focuses are mainly on specific tasks and objects. Coordinating and generalizing such modules to diverse common tasks, such as multiple object interactions in complex environments, remains a significant challenge.

Recently, the rapid advancement of Multimodal Large Language Models (MLLMs) (Li et al., 2022a; Alayrac et al., 2022; Liu et al., 2023c;e), such as GPT-4V (OpenAI, 2023b), have demonstrated strong aptitude in scene comprehension, understanding of objects' characteristics, and generalization capabilities across diverse scenes and objects. This suggests that leveraging their cognitive strengths by establishing connections to physical world interactions could enable more efficient intelligent system design. Additionally, while MLLMs excel at high-level scene understanding, robotic systems specialize in precise trajectory planning and execution. This implies that code could act as a symbolic bridge between conceptual knowledge and low-level behaviors for robots. Such connections would enable the integration of MLLMs' advanced cognitive processing with the nuanced requirements of robotic control, leading to more foundational and adaptable cross-platform robotic systems.

Inspired by the above observations, in this paper, we propose a large vision language model for robotic code generation, named RoboCodeX, to serve as an interface between MLLMs and robotic control systems, translating high-level semantics and physical preference into corresponding low-level motions tailored to a robot's mechanics. Critically, representing these plans and preferences in code enables sharing and transfer across morphologically distinct robots.

Specifically, RoboCodeX comprehends diverse sensory inputs like vision, depth data, and human instructions, and employs a tree-of-thought structure that decomposes high-level human instructions into several object-centric manipulation units defined through textual code annotations. Each unit acts as a node, further broken down into several key components: **1)** Target pose proposals generation through detailed 3D spatial analysis of objects within their environments. **2)** Object physical characteristic prediction with visual inspiration and 3D geometry information such as translation or rotation axis constraints on articulated objects, ensuring the generation of realistic and feasible motion plans. **3)** Preference prediction with the multimodal understanding of environmental and task objectives, selecting optimal gripping positions and approach directions. **4)** Trajectory generation leverages the planning algorithm to produce safe, optimized, and compliant motion plans to respect collision and physical constraints. With the seamlessly integrated key capabilities, RoboCodeX performs complex manipulation tasks flexibly in cluttered physical environments based on high-level commands.

Training an MLLM with such reasoning capacity for physical robotic preferences and robotic behavior in codes remains a challenge, which is often underrepresented in standard pretraining datasets primarily focusing on high-level visual and language alignment. Meanwhile, research on reasoning by fusing information from multiple sources is seldom explored. To address this gap, we construct a multimodal reasoning dataset for pretraining by generating simulated environments with diverse tasks and corresponding executable codes for each task. For the supervised fine-tuning (SFT) stage, we generate high-quality data and code with high success rate via an iterative iterative refining framework. The key contributions of this work are as follows:

- We introduce RoboCodeX, a large vision language model with tree-of-thought reasoning capabilities for robotic code generation. It predicts physical constraints, preferential rankings, and target position proposals while serving as an interface between high-level conceptual knowledge and low-level robotic behaviors.
- We present a specialized multimodal reasoning dataset and an iterative iterative refining methodology for supervised fine-tuning to enhance RoboCodeX's capacity for translating semantics and physical preferences into robot-specific motions.
- Extensive experiments demonstrate state-of-the-art performance of RoboCodeX in both simulated and real robot systems across four different kinds of manipulation tasks, *e.g.*, 17% success rate improvement over GPT-4V, and competitive performance on embodied navigation tasks.

2. Related Works

Code Generation for Robotic Control. Recent advances have explored utilizing natural language models for robotic behavior synthesis. ProgPrompt (Singh et al., 2023) introduced in the context of virtual environments, leverages natural language models to generate action sequences in VirtualHome (Puig et al., 2018), enabling intuitive and manageable robot behavior synthesis tailored for domestic settings. Building upon this, the Code-as-Policies (Liang et al., 2023) investigates composing full policy code for manipulation and navigation tasks by emphasizing few-shot prompting and hierarchical code generation. This work incorporates modular structures for reasoning and control into the generated programs. Further developments by (Vemprala et al., 2023) focus these techniques on real robotic systems. By mapping high-level functions to diverse atomic tasks, they rapidly adapt capabilities across robotic form factors while generating tailored executable code. RoboScript (Chen et al., 2024) further develop a benchmark for robotic code generation based on Robot Operating System. However, these methods rely solely on linguistic inputs, lacking multimodal grounding to map visual observations to behaviors. Different from previous works, this paper achieves visually-grounded code synthesis to generate behaviors specialized to perceived object characteristics. Remarkably, it can directly infer interactive preferences and physical constraints from observations, and generate executable code tailored to robotic systems.

Embodied AI with Large Foundation Models. Leveraging large pre-trained models shows promise for creating capable embodied agents. Many works focus on using language models for planning and reasoning in embodied agents (Huang et al., 2022a; Ahn et al., 2022; Chen et al., 2023a; Singh et al., 2023; Huang et al., 2023a; Raman et al., 2022; Song et al., 2023; Zhou et al., 2023a; Hu et al., 2023; Liu et al., 2023a; Vemprala et al., 2023; Ding et al., 2023; Driess et al., 2023; Yuan et al., 2023; Lu et al., 2023b; Wang et al., 2023a; Radford et al., 2019; Padalkar et al., 2023; Sha et al., 2023; Hu et al., 2023). To enable language models to perceive physical environments, common approaches include providing textual descriptions of scenes (Huang et al., 2022b; Zeng et al., 2022; Singh et al., 2023) or access to perception APIs (Liang et al., 2023). Vision can also be incorporated by decoding with visual context (Huang et al., 2023b) or using multi-modal language models that directly take visual input (Driess et al., 2023; OpenAI, 2023a; Mu et al., 2023b; Yang et al., 2023a; Mu et al., 2023a). Several researchers further adapt visual models to the embodied perspective under egocentric conditions (Nair et al., 2022; Huang et al., 2018) and build datasets for bridging the asynchronous egocentric and third-person view of human activities in the real world (Grauman et al., 2022; Huang et al., 2023c). However, perception alone is not enough for embodied agents - they must also know how to act. This is often achieved by providing pre-defined action primitives that the model can invoke. As shown in (Liang et al., 2023), LLMs can exhibit useful behavioral common sense for low-level control policies, in addition to high-level reasoning and planning capabilities. However, effectively connecting perception, reasoning, and grounded action primitives remains an open challenge. Unlike previous approaches that use separate pipelines for vision and language or only focus on high-level planning, our proposed method bridges the gap between the cognitive abilities of a large end-to-end multi-modal model and precise robotic planning through code generation. This approach allows the model to make multi-modal predictions, expanding each code node with information about target positions, physical properties, preferential rankings, and feasible trajectories.

Multimodal Large Language Models. Recent advancements have seen the creation of multimodal large language models (MLLMs) (Zhang et al., 2023c;b;d; Wu et al., 2023a; Sun et al., 2023; Alayrac et al., 2022; Zhu et al., 2023b; Li et al., 2023d; Lai et al., 2023; Yang et al., 2023b; Chen et al., 2022; Li et al., 2023a; Zhang et al., 2023a; Li et al., 2023f; Ye et al., 2023), which aim to enhance LLM with the capability to process and interpret visual information. Flamingo (Alayrac et al., 2022) uses the visual and language inputs as prompts and shows remarkable few-shot performance for visual question answering. Subsequently, LLaVA series (Liu et al., 2023d; Lu et al., 2023a; Liu et al., 2023c) and MiniGPT-4 (Zhu et al., 2023a) have brought in visual instruction tuning, to improve the instruction-following ability of MLLMs. Concurrently, models such as VisionLLM (Wang et al., 2023b), KOSMOS-2 (Peng et al., 2023), and Qwen-VL (Bai et al., 2023), along with other works (Wang et al., 2023c; Chen et al., 2023b) have improved MLLMs with visual grounding capabilities, facilitating tasks such as region description and localization. However, despite the considerable achievements of MLLMs, their multimodal reasoning capabilities in complex robotic behavior synthesizing remain under-explored.

3. Methods

We present a cutting-edge multi-modal code generation framework that employs a tree-of-thought architecture to effectively decompose instructions into actionable code units. This innovative framework is grounded in RoboCodeX, an advanced multi-modal vision-language model renowned for its versatility and comprehensiveness. RoboCodeX operates seamlessly by ingesting both image observations and human language instructions, processing them to derive the underlying thought process, and subsequently translating them into precise robot control codes. The foundation of RoboCodeX lies in its ability to comprehend complex

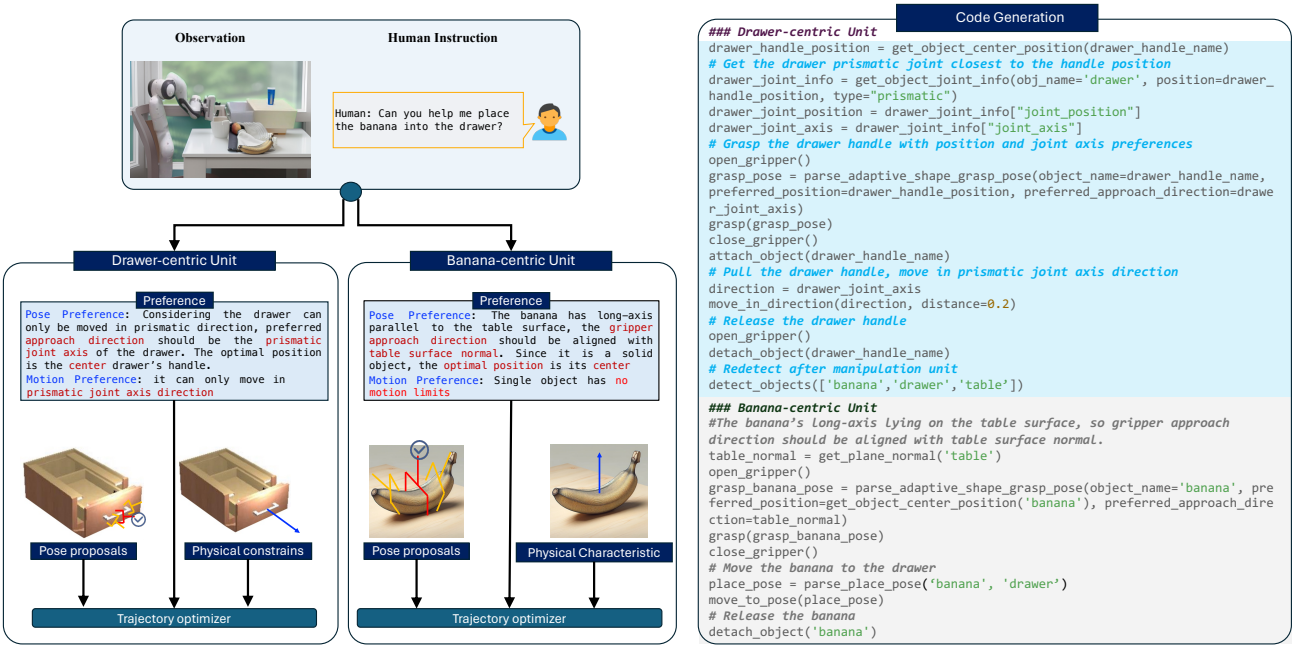

Figure 2. Example of robotic behavior synthesis with RoboCodeX. To accomplish the task "Place the banana into the drawer," we first decompose the whole task into a Drawer-centric Unit and a Banana-centric Unit. In the Drawer-centric Unit, the robot is programmed to understand that it must align its gripper with the prismatic joint axis of the drawer, which is the optimal position for movement, considering the drawer's physical limits and trajectory optimization. Conversely, the Banana-centric Unit requires the robot to align its gripper with the table surface normal and close to its center to pick up a banana. The accompanying code generation segment translates these multimodal considerations into executable instructions. For the drawer, the code includes determining the handle's position, executing the grip and pull actions in line with the drawer's joint axis, and then releasing the handle. For the banana, the code sequences involve aligning the gripper, grasping the banana, moving it to the drawer, and detaching it at the destination.

scenes, dividing tasks into distinct object-centric units. For each unit, it deduces interaction preferences and physical constraints based on the current visual observations before crafting the corresponding control codes. Throughout its operation, RoboCodeX engages with a myriad of proprietary API tools, including 2D grounding, grasp pose proposal, and articulated information prediction models. These tools enhance its capabilities, enabling it to perform tasks with unparalleled efficiency and accuracy.

3.1. Problem Setup

We consider a long-term manipulation problem as a high-level free-form human instruction $\mathcal{L}_{\text{global}}$ (*e.g.*, "clean the table"). The observation contains the RGBD data I from three different views of the depth camera (left, right, top). Since it is hard to generate the whole trajectories directly for long-term high-level instructions, we use a tree-of-thought method via the visual reasoning of MLLMs to decompose the entire task into several object-centric individual units $\mathcal{U}_{task} \rightarrow (u_1, u_2, \ldots, u_n)$. The central problem investigated in this work is how to generate a motion trajectory set $\{\tau_i\}_{i=0}^{n}$. We represent τ_i as a sequence of dense end-effector waypoints to be executed by an Operational Space Controller (Khatib, 1987), where each waypoint consists of a desired 6-DoF end-effector pose, end-effector velocity, and gripper action. Given the i-th sub-task described by ground-truth instruction ℓ_i^*, we formulate an optimization problem defined as follows:

$$\min_{\tau_i} \sum_{i=0}^{N} \{S_{\text{task}}\ (\tau_i, \ell_i^*) + S_{\text{control}}\ (\tau_i)\} \quad s.t. \quad \mathcal{C}(\tau_i), \qquad (1)$$

where S_{task} scores the extent to which τ_i completes the instruction ℓ_i, while S_{control} specifies the control costs, *e.g.*, the cost to encourage τ_i to minimize total control effort or total time. $\mathcal{C}(\tau_i)$ denotes the dynamics and kinematics constraints. By solving this optimization for each sub-task ℓ_i, we obtain a sequence of robot trajectories that collectively fulfill the overall task specified by the instruction $\mathcal{L}_{\text{global}}$.

3.2. Multi-modal Tree-of-thought Code Generation

We present a novel framework for synthesizing robotic behavior, integrating multi-perspective 3D perception, multi-modal chain-of-code reasoning, and motion planning for the execution of complex real-world tasks. The process begins by capturing observations I, comprising 3 RGBD frames from 3 different views. This approach helps avoid occlusion issues caused by a single viewpoint and provides comprehensive 3D information. These are fused into a unified

3D spatial representation, *i.e.*, a Truncated Signed Distance Field (TSDF), encoding precise environmental structure.

Concurrently, the RGB images from three views are fed as inputs into the RoboCodeX model together with the natural language instructions. RoboCodeX employs a tree-of-thought architecture centered around semantic parsing, contextual grounding, and goal-oriented partitioning. Such visual reasoning directly connects visual features to an understanding of interaction preferences and physical constraints on objects. The overall task is decomposed into sequential object-centric units as demonstrated in Equation (2) with the language description of the sub-task L_i and the manipulation preference pf_i which indicates the approach directions or preferred touching positions considering the control stability. For example, spherical objects should be grasped from the center for stability, while open containers like cups are better grasped at the edge for flexible control.

$$\begin{aligned} L_{\text{units}}, pf_{\text{units}} &= f\left(I, L_{\text{global}}\right) \\ L_{\text{units}} &= \{L_0, L_1, \ldots, L_N\} \\ pf_{\text{units}} &= \{pf_0, pf_1, \ldots, pf_N\}. \end{aligned} \tag{2}$$

For each object-centric unit, the grounded 2D positions derived from RGB streams are matched with 3D boxes from the point cloud, based on overlap and orientation consistency. This process enables the extraction of an accurate 3D point cloud, denoted as O_i for the task-relevant object. Subsequently, the object-centric trajectory generation problem, conditioned on the language instruction of the sub-task L_i can be formulated as,

$$\tau_i = g\left(O_i, L_i, pf_i\right), \tag{3}$$

where τ_i is the motion trajectory of the i-th sub-task generated by the function $g(\cdot)$ with the preference pf_i and language instruction L_i for the i-th task. It contains a sequence of dense end-effector waypoints to be executed and discrete control instructions for gripper opening and closing.

Then the aggregated object-specific perceptual inferences, physical insights, and manipulation parameters are systematically compiled into structured executable action code. Each unit is considered as a parent node and is further expanded to ***i) part-level affordance*** $A_i = h(O_i, L_i)$ prediction, which is the task-relevant part point cloud segmentation, such as the handle's area of the drawer; ***ii) grasp pose proposals*** pc_i prediction, $pc_i = \{pc_{i0}, \ldots, pc_{ij}, \ldots, pc_{ik}\} = \text{cand}\left(A_i\right)$. Function cand$(\cdot)$ is a mapping $\mathbb{N}^3 \rightarrow \text{SE}(3)$ to generates SE(3) poses. Specifically, we use pre-trained AnyGrasp (Fang et al., 2023) to generate the grasp pose proposals, which provide abundant candidates with scores (see details in Appendix F). We select the top 10 candidates as grasp pose proposals and use the preferences inferred by visual reasoning to select the optimal one, with consideration of the object's shape, prior knowledge, and the adjacent robot manipulation cells that will be performed. This leverages AnyGrasp's ability to produce diverse candidates while allowing task-specific selection via visual reasoning to pick high-quality ones; ***iii) object physical property*** prediction $\phi_i = \{\phi_{i0}, \ldots, \phi_{ij}, \ldots, \phi_{ik}\} = z(O_i, L_i)$ such as the joints information of the articulated objects. Specifically, we use the GAMMA (Yu et al., 2023b) model to predict the physical properties of articulated objects, which segments an articulated object point cloud into rigid parts and estimates articulation parameters. GAMMA extracts point-wise features using PointNet++ (Qi et al., 2017) which are then processed for segmentation, part offset, and joint axis regression (see details in Appendix E). We also encapsulate the plane detection functions in Open3D (Zhou et al., 2018) into an API for calling during code generation, taking the object's point cloud as input and outputting the detected planes as well as normal vector information; and ***iv) trajectory planning***: By integrating motion planning algorithms and the Robot Operating System (ROS) manipulation modules, the model finally outputs dynamically feasible robot trajectories with assurances of collision avoidance and singularity exclusion.

$$\tau_i = \mathcal{Z}(pc_i, \phi_i, pf_i, L_i). \tag{4}$$

Specifically, we employ zeroth-order optimization in trajectory planning by randomly sampling trajectories and evaluating them with S_{control} and the constraint from the occupancy map (see example in Appendix Figure 9).

3.3. Dataset Preparation

Pre-training Dataset. Most multi-modal models currently focus on aligning different modalities, leaving a research gap in complex reasoning with information from multiple sources. To address this, we develop a diverse robotic multi-modal code generation dataset with the help of GPT-4. To ensure the diversity of the dataset, we design a procedural data generation framework. We first randomly sample household scenes from the HM3D dataset (Ramakrishnan et al., 2021), which provides various indoor scenes like bedrooms, living rooms, and kitchens as the base environments. We then insert additional objects into semantically appropriate locations in the scenes—for example, placing objects on top of tables and counters. The inserted objects are of two types: independent objects like balls, toys, and fruits; and container objects like bowls, plates, and cups that can hold other objects. The objects are sampled from Google Scan Dataset (Downs et al., 2022), YCB Dataset (Calli et al., 2015), OmniObject3D Dataset (Wu et al., 2023b), and articulated object dataset AKB-48(Liu et al., 2022). By randomly selecting object categories and quantities and populating the scenes accordingly, complex scene configurations are obtained. Built upon the resulting environments, we form natural language descriptions which are inputted into the GPT-4 language model to produce free-form task descriptions suitable for the given scene configurations, specifying

goals like object manipulation and rearrangement. Note that since we can obtain all the information from the simulator and render the observation from any view, we can use GPT-4 to generate data rather than GPT-4V, which is more expensive for large-scale data generation. Finally, for each generated task, programming codes that accomplish assigned tasks are generated by GPT-4 from the task descriptions and additional parameterized inputs. We sample 10 high-quality code samples for each task which evaluated by GPT-3.5 and filter out those with syntax errors, obtaining executable programs for the robot to carry out in simulation. Through this pipeline combining environment generation, task specification, and program synthesis with large language models, we build a diverse dataset for pre-training robotic vision language reasoning models to accomplish various daily household tasks. It consists of 147,303 samples of multi-round conversations with image inputs. Notably, it includes 100 randomly generated scenarios, each with around 100 distinct robot tasks, such as "place the object into the drawer", "move object 1 0.5m away from object 2", "swap the position of object 1 and object 2", aimed at providing diverse scenarios for pretraining models in scene understanding, question answering, and robot task code generation. The sample lengths range from 1,649 to 2,446 tokens, with an average of 2,015.4 tokens. To avoid overfitting, we use both the general vision language pre-training dataset and the generated dataset together during the pre-training process. The general vision language pre-training dataset we use contains ShareGPT4V (Chen et al., 2023c) dataset, SViT (Zhao et al., 2023) dataset, and the LLaVA Visual Instruct 150K dataset (Liu et al., 2023d). During the pretraining stage, the RoboCodeX Pretrain Dataset was combined with other general VQA data (ShareGPT4V, SVIT, LLaVA-150K) in a 1:1 ratio.

SFT Dataset. The supervised fine-tuning (SFT) process (Dong et al., 2023) is meticulously designed to refine the model's proficiency in robotic code generation by training the model with a curated selection of high-quality code sequences that demonstrate a high success rate in evaluation. To create high-quality and diverse code data, we take the task types from the RT-1 (Brohan et al., 2022) and LIBERO (Liu et al., 2023b) datasets as a basis and randomly combine them with diverse objects to create a diverse set of tasks for each task type. For each type of task, we provide high-quality examples generated by humans, which have been verified to ensure successful completion in both simulator and real-world environments. We utilize GPT-4 to generate corresponding code for each task, taking high-quality examples and API explanations as input. For code with correct syntax but failed execution, we conduct an exhaustive search of optional settings, including grasping method (central lift or AnyGrasp), preferred contact position, preferred gripper direction, and trajectory generation method (fixed direction corresponding to objects' joints or only fixed final position). With the results of the search, we select the best option as the label and use GPT-4V to analyze why it is better and summarize it into coding annotations that serve as the label of the chain-of-thought. For the codes that maintain a zero success rate even after exploring optional settings, we manually revise the code and incorporate it into the example pool of human-labeled data. It contains 49,320 samples, focusing on high-quality multi-round conversations with image inputs and associated robot task code generation. This dataset is designed specifically for SFT process. The samples in the RoboCodeX SFT Dataset have a maximum length of 3,142 tokens and a minimum length of 1,928 tokens, with an average length of 2,259.8 tokens. The dataset includes only those samples where the generated robot code has achieved a success rate of over 50%, indicating semantic correctness and practical viability. It's important to note that during the SFT process, we still require general vision language data to prevent overfitting. Specifically, we utilize the same subset of the ShareGPT4V dataset used in the ShareGPT4V-7B model's SFT process (Chen et al., 2023c). During the subsequent supervised fine-tuning (SFT) stage, the RoboCodeX SFT Dataset was combined with other general VQA data at a ratio of 10:1.

3.4. Vision Language Model Design

Our model adopts the fundamental paradigm of BLIP2 (Li et al., 2023b), consisting of a vision transformer, Q-Former, and language model. As code generation for complex tasks tends to have considerable length and requires inputting long API documentation prompts, we leverage the Q-Former to bridge and compress the token quantity from the visual modality. Specifically, the vision transformer encodes visual elements into rich representational tokens. As full sequencing of these vision tokens alongside lengthy textual prompts may result in a high requirement of GPU memory, we adopt Q-Former to judiciously summarize the visual embeddings into more compact sequence tokens. These condensed representations capture the most task-relevant visual concepts while reducing sequence length to save GPU memory. The compressed Q-Former visual sequence tokens are then fed into the language model along with text tokens from code and documentation prompts. Furthermore, in order to obtain hierarchical features from the image, we design an efficient vision adapter that aggregates the features from different stages in the vision transformer. We first divide the hidden layers of the vision model into four parts proportionally. Then, the class token is extracted from the final layer of each part because it interacts with all tokens through self-attention within that layer. We concatenate these class tokens from the four hierarchical layers and enable interactions among them via the vision adapter. The vision adapter is a channel-wise attention network (Yan et al., 2021), which

first reduces the channel dimension through a linear layer, then selects features using a SILU (Paul et al., 2022) activation, and finally restores the original channel dimension using another linear layer (see pseudo-code in Appendix B). Finally, the aggregated feature tokens are concatenated with the other visual tokens. This combined set then serves as the input for the subsequent modules.

4. Experiments

4.1. Evaluation on Manipulation Task

We evaluate our framework on a series of robotic manipulation tasks with increasing complexity utilizing Gazebo (Qian et al., 2014) for the realistic rendering of the environment and the physical interactions: 1) open-vocabulary picking-and-place task which involves 42 different object categories from the YCB and Google Scanned object datasets, 2) Opening and closing drawers from 5 typical cabinets taken from PartNet-Mobility dataset (Xiang et al., 2020), 3) Opening and closing doors from the same set of 5 cabinets, 4) Placing objects in drawers, with the robot instructed to interact with the ordered drawer, and 5) Multi-stage tasks composed of sequential subtasks that need to be completed in order, such as putting all the fruits in the drawer(see detailed setup in Appendix A). The baselines consist of various multimodal models and large language models, including GPT-4V, GPT-3.5, and GPT-4, each equipped with open vocabulary object detection models. Among them, GPT-4V is a multimodal model taking images and language instructions as input, while GPT-4 and GPT-3.5 are language-only models, the inputs being text scene descriptions containing object names, attributes, and language instructions. All tested models share the same API and prompt. For open vocabulary detection, we choose GLIP-Large (Li et al., 2022b), owing to its flexibility in handling diversified objects like "tiny panda figure" in our tasks. We provide all the prompts and the explanation of the APIs for code generation in Appendix I.

As shown in Figure 3, RoboCodeX outperforms GPT-4V on pick-and-place tasks, primarily due to using more suitable priors for objects that GPT-4V struggled with. Compared to GPT-4, adding visual inputs enables GPT-4V to infer better grasp preferences. However, GPT-3.5 exhibited weaker reasoning abilities and often generated syntactically flawed text. As shown in Table 1, RoboCodeX demonstrates stronger capabilities than GPT-4V on articulated object manipulation and long-term tasks. Its core competency lies in locating target objects and corresponding handles across multiple drawers and doors in the cabinet, and properly operating them under physical constraints. For example, determining whether the target cabinet door should be rotated clockwise or counterclockwise relies more heavily on visual understanding, which highlights RoboCodeX's advanced competencies in robotic manipulation beyond text-based interfaces.

Table 1. Performance comparison among different tasks. We report the average success rate of all the tasks over 50 trials.

	Pick & Place	Drawers	Doors	Put obj. in drawer	Multi-stage
GPT-3.5	0.45	0.44	0.20	0.36	0.20
GPT-4	0.56	0.68	0.46	0.48	0.44
GPT-4V	0.63	0.68	0.52	0.55	0.50
RoboCodeX	**0.80**	**0.84**	**0.74**	**0.68**	**0.64**

Table 2. Performance on Embodied Navigation Tasks. We report the success rate and Success weighted by Path Length (SPL) as the evaluation metrics(Anderson et al., 2018).

Method	HM3D		HSSD	
	Success ↑	SPL ↑	Success ↑	SPL ↑
L3MVN(GPT2)	35.2	16.5	38.4	19.4
Pixel-Nav(GPT4)	37.9	20.5	-	-
ESC(GPT3.5)	39.2	22.3	-	-
RoboCodeX	**40.0**	**24.2**	**40.3**	**22.0**

Since long-term tasks involve many influencing factors, to gain a clear understanding of how much each factor plays into the whole long-term manipulation process, as shown in Figure 6, we specifically conducted error analyses on RoboCodeX's and GPT-4V's long-term task performance. The identification of errors relies on pinpointing specific stages of code execution that led to failure. These stages are derived from independently executing code segments. Each segment is automatically assessed to determine whether it successfully accomplishes its task or results in failure. The results show both models can correctly understand and decompose these tasks. The primary errors stem from sub-optimal grasp pose selection and trajectory planning failures, with the latter strongly correlated to grasp poses. Furthermore, RoboCodeX's multi-modal code generation training effectively reduces this error margin compared to GPT-4V.

4.2. Evaluation on Embodied Navigation Task

To test the performance of RoboCodeX on the visual language object-navigation task, we conduct experiments based on the HM3D (Ramakrishnan et al., 2021) and HSSD (Khanna et al., 2023) datasets. For navigation tasks, RoboCodeX primarily utilizes the L3MVN (Yu et al., 2023a) base framework, where it uses LLM to decide which frontier to select. The key difference is that RoboCodeX performs multimodal visual reasoning over the current observation and candidate frontiers to determine where to explore. We select the typical methods that use large foundation models to assist robots on visual language object navigation as key baselines, including L3MVN, Pixel-Nav (Cai et al., 2023) which analyzes panoramic images and utilizes GPT-4 to determine the optimal pixel for exploration, and ESC (Zhou et al., 2023b) which uses GPT-3.5 (Ouyang et al., 2022) to determine the mid-term goal from the frontier points during exploration. As shown in Table 2, compared to these baselines, our method achieves better performance and is

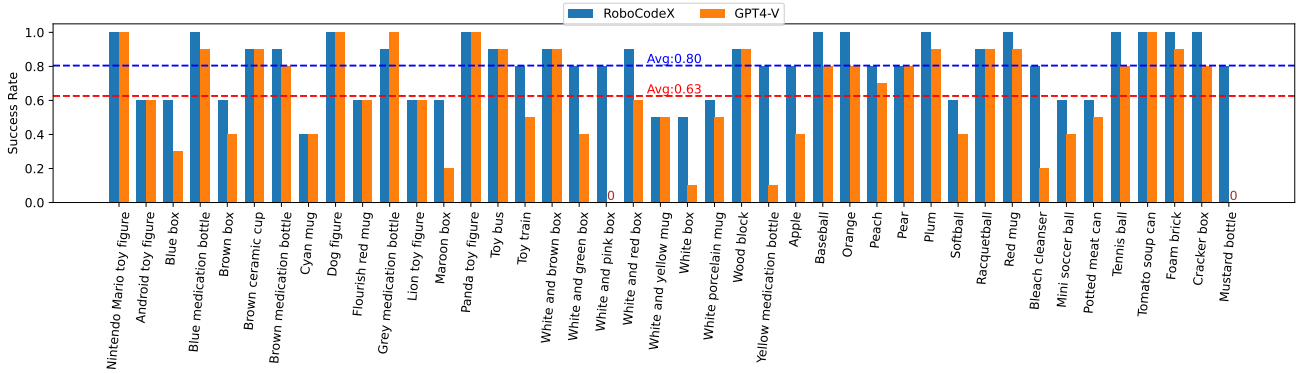

Figure 3. Performance Comparison on pick and place task with diverse objects.

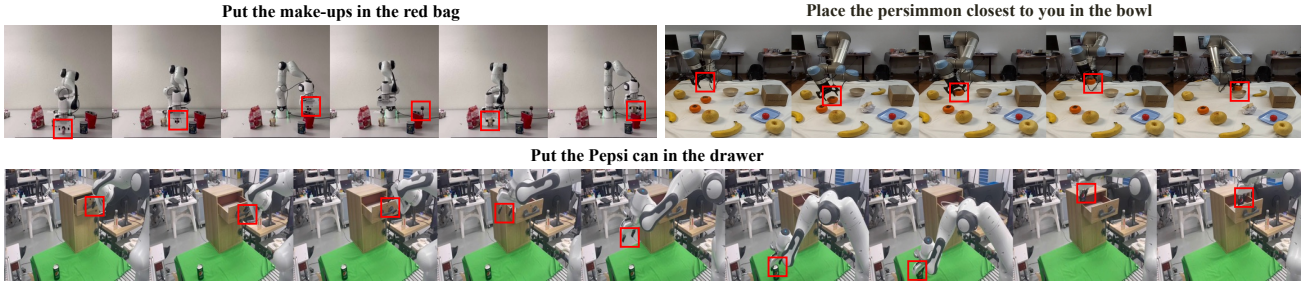

Figure 4. Generalization among different types of robots in real world without any fine-tuning. We evaluate RoboCodeX with Franka Emika Panda robot and UR5 robot in real world.

Table 3. Performance on general multimodal reasoning.

Model	LLaVA-Bench	MM-Vet
BLIP-2	38.1	22.4
InstructBLIP-7B	60.9	26.2
InstructBLIP-13B	58.2	25.6
LLaVA-1.5-7B	63.4	30.5
LLaVA-1.5-13B	70.7	**35.4**
RoboCodeX-13B	**71.5**	31.0

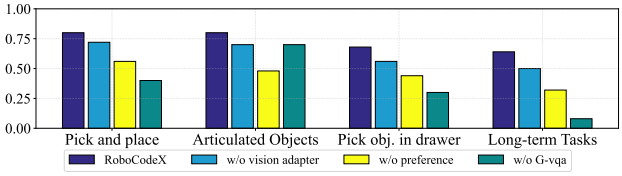

Figure 5. Ablation on the utilization of preference, vision adapter, and whether to use general VQA data during fine-tuning. We report the average success rate over 4 kinds of tasks over 50 trials.

comparable to GPT-3.5, demonstrating its general visual reasoning ability. The key advancement lies in the ability of RoboCodeX to conduct multimodal visual reasoning, improving scene comprehension and logical reasoning.

4.3. Evaluation on General VQA

To test RoboCodeX's general multimodal understanding and reasoning capabilities, we utilize two benchmarks: 1) LLaVA-Bench (Liu et al., 2023d), a widely used benchmark for assessing multimodal conversation abilities. It scores

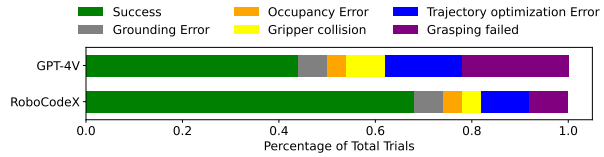

Figure 6. Failure modes comparison between RoboCodeX and GPT-4V in long-term tasks.

model responses against reference responses using GPT-4. 2) MM-Vet (Yu et al., 2023c), an evaluation benchmark focused on complicated multimodal tasks, designed with the insight that the ability to solve complex tasks is often achieved by integrating different core vision-language capabilities into a generalist model. As shown in Table 3, RoboCodeX demonstrates comparable general visual reasoning to LLaVA-1.5-13B (Liu et al., 2023c), the current state-of-the-art 13B multi-modal model, even without specialized fine-tuning on robotic tasks.

4.4. Real World Experiments

To validate the generalization ability of RoboCodeX to different real-world scenarios across different robot platforms, we test it on both the Franka Emika robot arm (Haddadin et al., 2022) and the UR5 robot arm (Kebria et al., 2016) which include multi-stage pick-and-place tasks, and put-object-in-drawer tasks, in a zero-shot way. As shown in Figure 4, the results show that our framework can adapt to different robots simply by changing the robot's configu-

ration file, demonstrating its strong generalization to new scenarios without any specific fine-tuning. See more quantitative results in Appendix H.

4.5. Ablation Study

We conduct ablation studies to evaluate the contributions of various components in the RoboCodeX's framework, which focus on examining the utility of the preference model, the vision adapter, and the incorporation of general visual question answering (G-VQA) data, , encompassing datasets such as ShareGPT4V (Chen et al., 2023c), SVIT (Zhao et al., 2023), and LLaVA-150K (Liu et al., 2023d).

Ablation on the preference. We evaluate whether the inferred preference is necessary for robust manipulation, compared to directly using the highest-scoring grasp predicted by Anygrasp. As shown in Figure 5, the preference significantly outperforms the baseline across all the tasks, which highlights its efficacy in enhancing manipulation stability and aligning with subsequent planning.

Ablation on vision adapter. To evaluate the effectiveness of the vision adapter, we conduct an experiment by removing it from the model. The results indicate that the vision adapter contributes to improving the success rate, primarily by aiding the model in better understanding the fine details of various objects. However, while it enhances performance, it is not the most critical component that determines the overall performance of the model.

Ablation on general VQA data utilization. We explore the necessity of leveraging general VQA data during the SFT process. The results indicate that without general VQA (w/o G-VQA) data, the model exhibits significant overfitting. Its ability to follow instructions for different objects has deteriorated and it often answers questions improperly, therefore the average success rate has dropped significantly.

Overall, the ablation studies confirm the critical role of preference prediction and the vision adapter. These core components work together to enable precise object handling and adaptation to a variety of tasks. Furthermore, the incorporation of general VQA data during fine-tuning is essential to prevent overfitting issues.

5. Limitation

While RoboCodeX has advanced the state-of-the-art in several robot manipulation tasks, it still faces significant limitations in handling tasks that require high adaptability and dexterous operations. For instance, it lacks optimal force sensor handling for precise assembly tasks like working with nuts and bolts. Moreover, it faces challenges with unstructured tasks such as wiping tables or sweeping, which demand a fluid and adaptable approach.

6. Conclusion

This work introduces RoboCodeX, a pioneering multimodal code generation framework that bridges the gap between multi-modal large language models (MLLMs) and robotic control systems, innovatively translating semantic understanding into tailored robotic behaviors. Through extensive experimentation, RoboCodeX demonstrates state-of-the-art performance in complex robotic manipulation tasks, showcasing its robustness and adaptability in both simulated and real-world environments. The integration of a multimodal tree-of-thought approach, specialized dataset, and iterative fine-tuning methodology significantly enhances the model's capacity to interpret visual observation and human instructions into precise, robot-specific actions. Our findings suggest that the fusion of MLLMs' cognitive strengths with nuanced physical world interactions heralds a new era in embodied AI, where robots can efficiently adapt to and manipulate their environment with unprecedented sophistication. Future research could explore the expansion of RoboCodeX's capabilities to more diverse tasks, further unlocking the potential of multi-modal AI in robotics.

Impact Statement

The proposed RoboCodeX framework represents a notable improvement in embodied AI and intelligent robotics by merging vision-language models with robotic control. This integration enables the translation of conceptual knowledge into precise physical actions across diverse platforms, offering potential enhancements in industrial, commercial, and domestic robotics. RoboCodeX has demonstrated strong performance in both simulated and real-world settings, underscoring the potential of cognitive-physical AI systems for tasks that were previously challenging for robots.

Acknowledgement

This paper is partially supported by the National Key R&D Program of China No.2022ZD0161000. We thank Professor Yufeng Yue and Dr. Guangyan Chen from the Beijing Institute of Technology for their expertise and insightful feedback. Special thanks to Dr. Yifei Huang from the Shanghai AI Lab for his support with egocentric human-object interaction data and model pretraining suggestions. We also appreciate the assistance of Professor Cewu Lu, Dr. Wenhai Liu, and Mr. Chenxi Wang from Shanghai Jiao Tong University in integrating the Anygrasp model into our system. We greatly appreciate the support from Professors Huazhe Xu and Xueqian Wang at Tsinghua University for their significant contributions in robot hardware, which have greatly enhanced our project. We also thank Dr. René Zurbrügg from the ETH AI Center for his help with our experimental platform.

References

Agrawal, A., Prabhakar, R., Goyal, A., and Liu, D. Physical reasoning and object planning for household embodied agents. *arXiv preprint arXiv:2311.13577*, 2023.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35: 23716–23736, 2022.

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Bai, J., Bai, S., Yang, S., Wang, S., Tan, S., Wang, P., Lin, J., Zhou, C., and Zhou, J. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Cai, W., Huang, S., Cheng, G., Long, Y., Gao, P., Sun, C., and Dong, H. Bridging zero-shot object navigation and foundation models through pixel-guided navigation skill. *arXiv preprint arXiv:2309.10309*, 2023.

Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. *arXiv preprint arXiv:1502.03143*, 2015.

Chen, B., Xia, F., Ichter, B., Rao, K., Gopalakrishnan, K., Ryoo, M. S., Stone, A., and Kappler, D. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11509–11522. IEEE, 2023a.

Chen, J., Mu, Y., Yu, Q., Wei, T., Wu, S., Yuan, Z., Liang, Z., Yang, C., Zhang, K., Shao, W., et al. Roboscript: Code generation for free-form manipulation tasks across real and simulation. *arXiv preprint arXiv:2402.14623*, 2024.

Chen, K., Zhang, Z., Zeng, W., Zhang, R., Zhu, F., and Zhao, R. Shikra: Unleashing multimodal llm's referential dialogue magic. *arXiv preprint arXiv:2306.15195*, 2023b.

Chen, L., Li, J., Dong, X., Zhang, P., He, C., Wang, J., Zhao, F., and Lin, D. Sharegpt4v: Improving large multi-modal models with better captions. *arXiv preprint arXiv:2311.12793*, 2023c.

Chen, X., Wang, X., Changpinyo, S., Piergiovanni, A., Padlewski, P., Salz, D., Goodman, S., Grycner, A., Mustafa, B., Beyer, L., et al. Pali: A jointly-scaled multilingual language-image model. In *ICLR*, 2022.

Dasari, S., Gupta, A., and Kumar, V. Learning dexterous manipulation from exemplar object trajectories and pre-grasps. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3889–3896. IEEE, 2023.

Ding, Y., Zhang, X., Paxton, C., and Zhang, S. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

Dong, G., Yuan, H., Lu, K., Li, C., Xue, M., Liu, D., Wang, W., Yuan, Z., Zhou, C., and Zhou, J. How abilities in large language models are affected by supervised fine-tuning data composition. *arXiv preprint arXiv:2310.05492*, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

Downs, L., Francis, A., Koenig, N., Kinman, B., Hickman, R., Reymann, K., McHugh, T. B., and Vanhoucke, V. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2553–2560. IEEE, 2022.

Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., Zeng, A., Mordatch, I., and Florence, P. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

Eisner, B., Zhang, H., and Held, D. Flowbot3d: Learning 3d articulation flow to manipulate articulated objects. *arXiv preprint arXiv:2205.04382*, 2022.

Fang, H.-S., Wang, C., Fang, H., Gou, M., Liu, J., Yan, H., Liu, W., Xie, Y., and Lu, C. Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 2023.

Fang, Y., Wang, W., Xie, B., Sun, Q., Wu, L., Wang, X., Huang, T., Wang, X., and Cao, Y. Eva: Exploring the limits of masked visual representation learning at scale. *arXiv preprint arXiv:2211.07636*, 2022.

Gao, J., Sarkar, B., Xia, F., Xiao, T., Wu, J., Ichter, B., Majumdar, A., and Sadigh, D. Physically grounded vision-language models for robotic manipulation. *arXiv preprint arXiv:2309.02561*, 2023.

Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18995–19012, 2022.

Haddadin, S., Parusel, S., Johannsmeier, L., Golz, S., Gabl, S., Walch, F., Sabaghian, M., Jähne, C., Hausperger, L., and Haddadin, S. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29(2):46–64, 2022.

Hu, M., Mu, Y., Yu, X., Ding, M., Wu, S., Shao, W., Chen, Q., Wang, B., Qiao, Y., and Luo, P. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*, 2023.

Huang, C., Mees, O., Zeng, A., and Burgard, W. Visual language maps for robot navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10608–10615. IEEE, 2023a.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022a.

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

Huang, W., Xia, F., Shah, D., Driess, D., Zeng, A., Lu, Y., Florence, P., Mordatch, I., Levine, S., Hausman, K., et al. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*, 2023b.

Huang, Y., Cai, M., Li, Z., and Sato, Y. Predicting gaze in egocentric video by learning task-dependent attention transition. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 754–769, 2018.

Huang, Y., Chen, G., Xu, J., Zhang, M., Yang, L., Pei, B., Zhang, H., Dong, L., Wang, Y., Wang, L., and Qiao, Y. Egobridge: A dataset for bridging asynchronous first- and third-person views of activities in the real world. 2023c. URL https://egobridge.github.io/static/videos/egobridge_paper.pdf.

Kebria, P. M., Al-Wais, S., Abdi, H., and Nahavandi, S. Kinematic and dynamic modelling of ur5 manipulator. In *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 004229–004234. IEEE, 2016.

Khanna, M., Mao, Y., Jiang, H., Haresh, S., Shacklett, B., Batra, D., Clegg, A., Undersander, E., Chang, A. X., and Savva, M. Habitat synthetic scenes dataset (hssd-200): An analysis of 3d scene scale and realism tradeoffs for objectgoal navigation, 2023.

Khatib, O. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

Koubâa, A. et al. *Robot Operating System (ROS).*, volume 1. Springer, 2017.

Lai, X., Tian, Z., Chen, Y., Li, Y., Yuan, Y., Liu, S., and Jia, J. Lisa: Reasoning segmentation via large language model. *arXiv preprint arXiv:2308.00692*, 2023.

Li, B., Zhang, Y., Chen, L., Wang, J., Yang, J., and Liu, Z. Otter: A multi-modal model with in-context instruction tuning. *arXiv preprint arXiv:2305.03726*, 2023a.

Li, J., Li, D., Xiong, C., and Hoi, S. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, pp. 12888–12900. PMLR, 2022a.

Li, J., Li, D., Savarese, S., and Hoi, S. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023b.

Li, J., Li, D., Savarese, S., and Hoi, S. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023c.

Li, K., He, Y., Wang, Y., Li, Y., Wang, W., Luo, P., Wang, Y., Wang, L., and Qiao, Y. Videochat: Chat-centric video understanding. *arXiv preprint arXiv:2305.06355*, 2023d.

Li, L. H., Zhang, P., Zhang, H., Yang, J., Li, C., Zhong, Y., Wang, L., Yuan, L., Zhang, L., Hwang, J.-N., et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10965–10975, 2022b.

Li, X., Zhang, M., Geng, Y., Geng, H., Long, Y., Shen, Y., Zhang, R., Liu, J., and Dong, H. Manipllm: Embodied multimodal large language model for object-centric robotic manipulation. *arXiv preprint arXiv:2312.16217*, 2023e.

Li, Z., Yang, B., Liu, Q., Ma, Z., Zhang, S., Yang, J., Sun, Y., Liu, Y., and Bai, X. Monkey: Image resolution and text label are important things for large multi-modal models. *arXiv preprint arXiv:2311.06607*, 2023f.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as Policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., and Stone, P. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Liu, B., Zhu, Y., Gao, C., Feng, Y., Liu, Q., Zhu, Y., and Stone, P. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023b.

Liu, H., Li, C., Li, Y., and Lee, Y. J. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*, 2023c.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *NeurIPS*, 2023d.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023e.

Liu, L., Xu, W., Fu, H., Qian, S., Yu, Q., Han, Y., and Lu, C. Akb-48: A real-world articulated object knowledge base. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14809–14818, 2022.

Lu, Y., Li, C., Liu, H., Yang, J., Gao, J., and Shen, Y. An empirical study of scaling instruct-tuned large multimodal models. *arXiv preprint arXiv:2309.09958*, 2023a.

Lu, Y., Lu, P., Chen, Z., Zhu, W., Wang, X. E., and Wang, W. Y. Multimodal procedural planning via dual text-image prompting. *arXiv preprint arXiv:2305.01795*, 2023b.

Mirjalili, R., Krawez, M., Silenzi, S., Blei, Y., and Burgard, W. Lan-grasp: Using large language models for semantic object grasping. *arXiv preprint arXiv:2310.05239*, 2023.

Mo, K., Zhu, S., Chang, A. X., Yi, L., Tripathi, S., Guibas, L. J., and Su, H. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Mu, Y., Yao, S., Ding, M., Luo, P., and Gan, C. Ec2: Emergent communication for embodied control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6704–6714, 2023a.

Mu, Y., Zhang, Q., Hu, M., Wang, W., Ding, M., Jin, J., Wang, B., Dai, J., Qiao, Y., and Luo, P. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *arXiv preprint arXiv:2305.15021*, 2023b.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023a.

OpenAI. Gpt-4 technical report, 2023b.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Padalkar, A., Pooley, A., Jain, A., Bewley, A., Herzog, A., Irpan, A., Khazatsky, A., Rai, A., Singh, A., Brohan, A., et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

Paul, A., Bandyopadhyay, R., Yoon, J. H., Geem, Z. W., and Sarkar, R. Sinlu: Sinu-sigmoidal linear unit. *Mathematics*, 10(3):337, 2022.

Peng, Z., Wang, W., Dong, L., Hao, Y., Huang, S., Ma, S., and Wei, F. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*, 2023.

Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., and Torralba, A. VirtualHome: Simulating household activities via programs. In *CVPR*, pp. 8494–8502, 2018.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., Deng, H., Hu, Y., and Zhang, J. Manipulation task simulation using ros and gazebo. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pp. 2594–2598. IEEE, 2014.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ramakrishnan, S. K., Gokaslan, A., Wijmans, E., Maksymets, O., Clegg, A., Turner, J., Undersander, E., Galuba, W., Westbury, A., Chang, A. X., et al. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv preprint arXiv:2109.08238*, 2021.

Raman, S. S., Cohen, V., Rosen, E., Idrees, I., Paulius, D., and Tellex, S. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Rana, K., Haviland, J., Garg, S., Abou-Chakra, J., Reid, I., and Suenderhauf, N. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. *arXiv preprint arXiv:2307.06135*, 2023.

Sha, H., Mu, Y., Jiang, Y., Chen, L., Xu, C., Luo, P., Li, S. E., Tomizuka, M., Zhan, W., and Ding, M. Languagempc: Large language models as decision makers for autonomous driving. *arXiv preprint arXiv:2310.03026*, 2023.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., and Garg, A. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and Su, Y. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009, 2023.

Sucan, I. A., Moll, M., and Kavraki, L. E. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19 (4):72–82, 2012.

Sun, Q., Yu, Q., Cui, Y., Zhang, F., Zhang, X., Wang, Y., Gao, H., Liu, J., Huang, T., and Wang, X. Generative pretraining in multimodality. *arXiv preprint arXiv:2307.05222*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vemprala, S., Bonatti, R., Bucker, A., and Kapoor, A. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res*, 2:20, 2023.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Wang, W., Chen, Z., Chen, X., Wu, J., Zhu, X., Zeng, G., Luo, P., Lu, T., Zhou, J., Qiao, Y., et al. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *NeurIPS*, 2023b.

Wang, W., Shi, M., Li, Q., Wang, W., Huang, Z., Xing, L., Chen, Z., Li, H., Zhu, X., Cao, Z., et al. The all-seeing project: Towards panoptic visual recognition and understanding of the open world. *arXiv preprint arXiv:2308.01907*, 2023c.

Wu, S., Fei, H., Qu, L., Ji, W., and Chua, T.-S. Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*, 2023a.

Wu, T., Zhang, J., Fu, X., Wang, Y., Ren, J., Pan, L., Wu, W., Yang, L., Wang, J., Qian, C., et al. Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 803–814, 2023b.

Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020.

Xu, Z., He, Z., and Song, S. Universal manipulation policy network for articulated objects. *IEEE Robotics and Automation Letters*, 7(2):2447–2454, 2022.

Yan, J., Zhao, H., Bu, P., and Jin, Y. Channel-wise attention-based network for self-supervised monocular depth estimation. In *2021 International Conference on 3D vision (3DV)*, pp. 464–473. IEEE, 2021.

Yang, J., Dong, Y., Liu, S., Li, B., Wang, Z., Jiang, C., Tan, H., Kang, J., Zhang, Y., Zhou, K., et al. Octopus: Embodied vision-language programmer from environmental feedback. *arXiv preprint arXiv:2310.08588*, 2023a.

Yang, Z., Li, L., Lin, K., Wang, J., Lin, C.-C., Liu, Z., and Wang, L. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 9, 2023b.

Ye, J., Hu, A., Xu, H., Ye, Q., Yan, M., Dan, Y., Zhao, C., Xu, G., Li, C., Tian, J., Qi, Q., Zhang, J., and Huang, F. mplug-docowl: Modularized multimodal large language model for document understanding, 2023.

Yu, B., Kasaei, H., and Cao, M. L3mvn: Leveraging large language models for visual target navigation. *arXiv preprint arXiv:2304.05501*, 2023a.

Yu, Q., Wang, J., Liu, W., Hao, C., Liu, L., Shao, L., Wang, W., and Lu, C. Gamma: Generalizable articulation modeling and manipulation for articulated objects. *arXiv preprint arXiv:2309.16264*, 2023b.

Yu, W., Yang, Z., Li, L., Wang, J., Lin, K., Liu, Z., Wang, X., and Wang, L. Mm-vet: Evaluating large multimodal models for integrated capabilities. *arXiv preprint arXiv:2308.02490*, 2023c.

Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., and Lu, Z. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023.

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

Zhang, H., Li, X., and Bing, L. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*, 2023a.

Zhang, P., Wang, X. D. B., Cao, Y., Xu, C., Ouyang, L., Zhao, Z., Ding, S., Zhang, S., Duan, H., Yan, H., et al. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112*, 2023b.

Zhang, R., Han, J., Zhou, A., Hu, X., Yan, S., Lu, P., Li, H., Gao, P., and Qiao, Y. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023c.

Zhang, S., Sun, P., Chen, S., Xiao, M., Shao, W., Zhang, W., Chen, K., and Luo, P. Gpt4roi: Instruction tuning large language model on region-of-interest. *arXiv preprint arXiv:2307.03601*, 2023d.

Zhao, B., Wu, B., and Huang, T. Svit: Scaling up visual instruction tuning. *arXiv preprint arXiv:2307.04087*, 2023.

Zhong, C., Zheng, Y., Zheng, Y., Zhao, H., Yi, L., Mu, X., Wang, L., Li, P., Zhou, G., Yang, C., et al. 3d implicit transporter for temporally consistent keypoint discovery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3869–3880, 2023.

Zhou, H., Ding, M., Peng, W., Tomizuka, M., Shao, L., and Gan, C. Generalizable long-horizon manipulations with large language models. *arXiv preprint arXiv:2310.02264*, 2023a.

Zhou, K., Zheng, K., Pryor, C., Shen, Y., Jin, H., Getoor, L., and Wang, X. E. Esc: Exploration with soft commonsense constraints for zero-shot object navigation. *arXiv preprint arXiv:2301.13166*, 2023b.

Zhou, Q.-Y., Park, J., and Koltun, V. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.

Zhu, D., Chen, J., Shen, X., Li, X., and Elhoseiny, M. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023a.

Zhu, X., Chen, Y., Tian, H., Tao, C., Su, W., Yang, C., Huang, G., Li, B., Lu, L., Wang, X., 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*, 2023b.

A. Manipulation Simulation Setup

In this section, we delineate the methodology employed for constructing a sophisticated robotic simulation, integrating a Franka robotic arm within a dynamic environment composed of different cabinets sourced from the PartNet-Mobility Dataset and a table adorned with various objects.

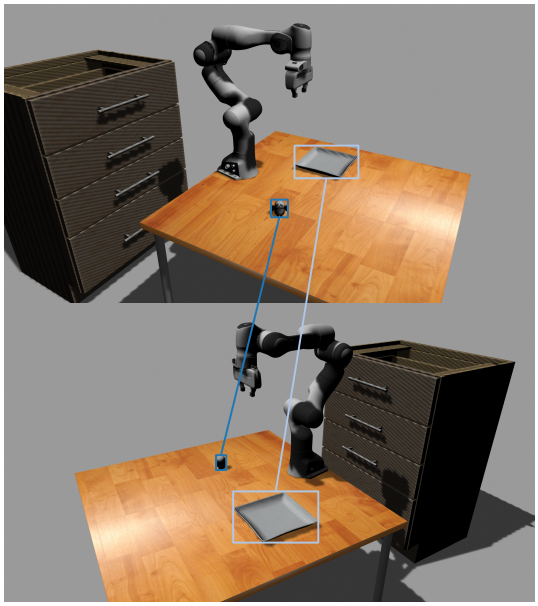

Figure 7. Matching results of 2D bounding boxes.

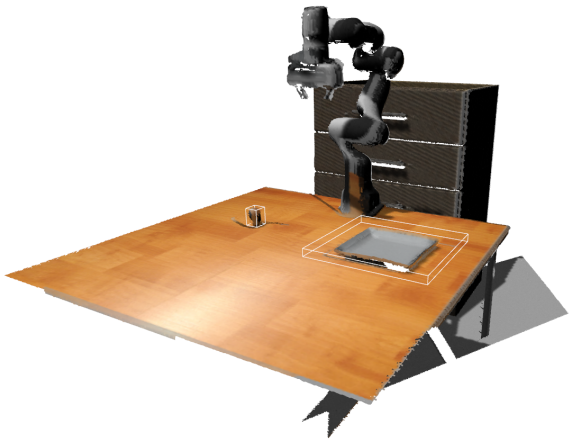

Figure 8. 3D bounding boxes.

The simulation framework is anchored in ROS(Koubâa et al., 2017) (Robot Operating System), utilizing Gazebo(Qian et al., 2014) for the realistic rendering of the environment and the physical interactions therein. The robotic arm's motion planning is facilitated by the integration of the MoveIt module, renowned for its comprehensive motion planning capabilities, and the OMPL(Sucan et al., 2012) (Open Motion Planning Library), which offers a suite of advanced algorithms for efficient path planning and obstacle avoidance. The choice of the Franka arm's URDF (Unified Robot Description Format) model and the incorporation of the PartNet-Mobility Dataset's cabinet model necessitated a meticulous conversion and scaling process to ensure compatibility with the Gazebo environment. This setup's fidelity and functionality were rigorously validated through iterative testing phases, focusing on the robot's ability to physically interact with the diverse objects. The agent processes three distinct RGBD images, utilizing the GLIP model for visual grounding, complemented by the reasoning outputs from large foundation models in our baseline. As shown in Figure 7, multi-view 2D bounding boxes are aligned using geometric correspondences derived from depth data. These aligned bounding boxes are then fed into the 3D perception loop. It integrates multi-view RGBD images and 2D perception grounding results to generate 3D bounding boxes for objects as shown in Figure 8, as well as their corresponding point clouds and occupancy maps as shown in Figure 9. Subsequently, the 3D point clouds are input into a grasp detection module, which is responsible for generating the grasp pose.

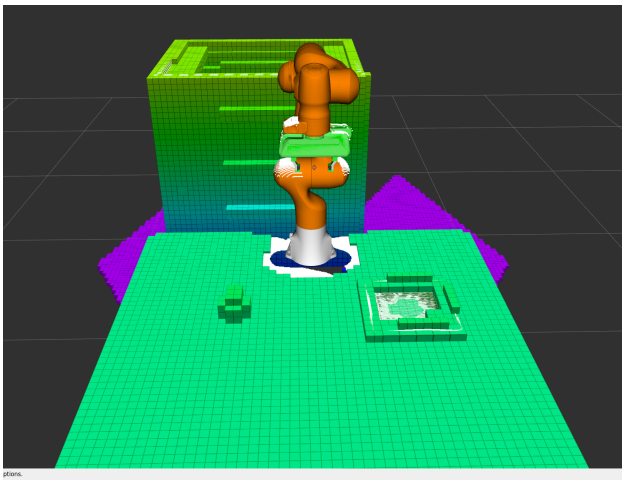

Figure 9. Visualization of the occupancy map. Occupancy maps represent the environment as a grid, where each cell can be free, occupied, or unknown.

B. Implementation details of Vision Langauge Model

We employ the same architecture as BLIP-2 (Li et al., 2023b). Specifically, we leverage the pre-trained ViT-G/14 model from EVA-CLIP (Fang et al., 2022) , modifying it by discarding its final layer and utilizing the output features from the penultimate layer instead. For our language model, we employ the pre-trained LLaMA-13B model(Touvron et al., 2023). The vision adapter undergoes

zero-initialization. We commence pretraining the entire vision-language model on a pretraining dataset inclusive of general Visual Question Answering (VQA) data and generated multi-modal code generation data. Following this, we fine-tune the complete vision-language model using the supervised SFT dataset, achieving a notably high success rate. We then utilize the obtained language model as the initial language model for multi-modal code-generation pretraining and special finetuning. Additionally, we convert the data type of parameters of the frozen ViT (Dosovitskiy et al., 2020) and language model to FP16 during pre-training to increase efficiency.

Configuration Component	Details
Model Type	RoboCodeX
Parameters	
- Hidden Size	5120
- Attention Heads	40 (Text), 16 (Vision)
- Intermediate Size	13824 (Text), 6144 (Vision)
Q-former	Standard Transformer
- Dropout Prob	0.1
- Encoder Hidden Size	1408
- Hidden Size	768
- Intermediate Size	3072
- Attention Heads	12
Vision Model	ViT
- Hidden Size	1408
- Intermediate Size	6144
- Attention Heads	16
Text Model	LLaMa
- Hidden Size	5120
- Intermediate Size	13824
- Attention Heads	40

Table 4. Model card of RobocodeX.

The detailed implementation of the vision adapter, which takes the concatenated class tokens from different stage in vision transformer as input, is demonstrated in Listing 1.

Listing 1. Pseudo-code of the Vision adapter

```
class Adapter(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.activation_fn = ACT2FN["silu"]
        hidden_size = config.vision_config.hidden_size
        intermediate_size = hidden_size // 4
        output_size = config.qformer_config.hidden_size
        self.fc1 = nn.Linear(hidden_size, intermediate_size)
        self.fc2 = nn.Linear(intermediate_size, output_size)
        self.layernorm = nn.LayerNorm(output_size, eps=config.
            vision_config.layer_norm_eps)
    def forward(self, hidden_states: torch.Tensor) -> torch.
        Tensor:
        hidden_states = self.fc1(hidden_states)
        hidden_states = self.activation_fn(hidden_states)
        hidden_states = self.fc2(hidden_states)
        hidden_states = self.layernorm(hidden_states)
        return hidden_states
```

C. Details of Dataset Collection

The dataset generation process aims to create two types of data for model training:

1. **Pre-training Data:** This data covers diverse scenes and tasks to facilitate pre-training of the model. The primary focus is on diversity, and the quality requirements are relatively relaxed; the system only filters out code that fails to compile, without necessarily ensuring successful task execution. Conducting comprehensive robot testing for vast amounts of data would be impractical.

2. **Data for Supervised Fine-Tuning (SFT):** The core requirement for this data is high code quality, with a higher success rate for executed tasks. This type of data requires iterative testing and validation.

Therefore, the data collection pipeline is mainly divided into two parts. The first part is mainly for the pre-training stage of data collection, with the main purpose of enhancing the model's ability to handle diverse scenarios and diverse robot tasks. As shown in Figure 10, the main steps include: 1) Randomly combining different scenarios from the model library 2) Generating different robot tasks for each scenario 3) Generating candidates for the corresponding control code for each task 4) Checking the syntactic validity of the generated data 5) GPT-4 itself selects the best option from the syntactically correct candidates as the task label.

The second part is mainly for data collection in the SFT stage, with the main purpose of collecting high-quality code generation data. As shown in Figure 11, the main steps include: 1) Code generation under the prompts from the human example code library 2) Syntax validity check 3) Simulation environment execution verification 4) GPT-4 automatically modifies preference options and other parts of the code based on execution feedback 5) For tasks with a still 0 success rate, manual revision is performed.

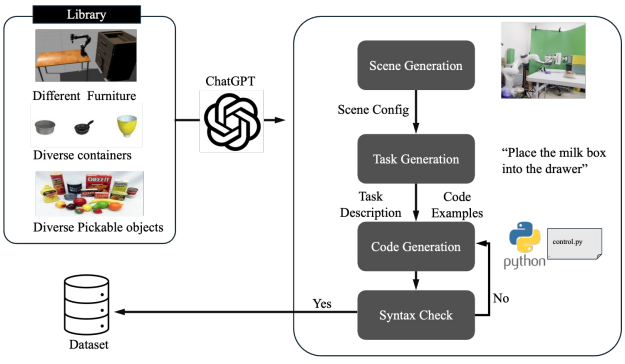

Figure 10. Collection process of RoboCodeX pre-training dataset.

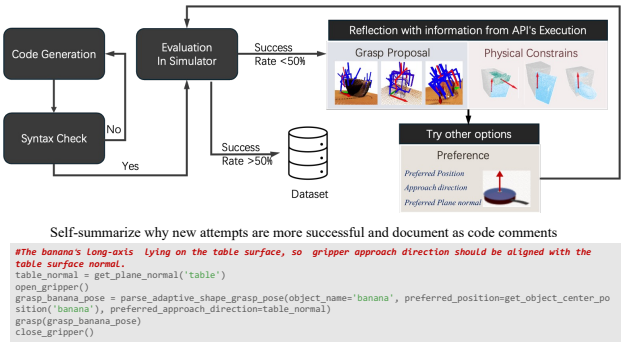

Figure 11. Collection process of RoboCodeX SFT dataset.

D. Explanation of the relationship between different components of RoboCodeX

RoboCodex is a standard vision-language multi-modal model, consisting of components such as the Vision Transformer, Vision Adapter, Q-former, and Language Model. It takes image observations and human instructions as input and outputs the thought process and generated code in text form. The generated code calls various predefined tool APIs, which are composed of other networks (such as AnyGrasp) or non-neural planning algorithms. The API functions called in the code and the respective inputs and outputs of the API functions are defined in the Appendix I. The Vision Transformer and Language Model components are initialized with pre-trained weights, the Q-former is initialized with pre-trained weights from BLIP2(Li et al., 2023c), and the Vision Adapter is zero-initialized. The whole multi-modal foundation model is then updated with two training stages, using the RoboCodeX pretraining dataset and SFT dataset respectively.

The 2D grounding and grasp pose proposal tools are predefined without any special training and are not involved in the training process of the RoboCodeX-13B model. Instead, they are called via APIs during RoboCodeX inference to provide additional information. The articulated information prediction model is fine-tuned to fit the specific camera setup used in the RoboCodeX system, which utilizes three depth cameras to reconstruct the point cloud, which is denser than the original GAMMA(Yu et al., 2023b) model. All the predefined tools, including 2D grounding, grasp pose proposal, and articulated information prediction tools, are not directly involved in the training process of the RoboCodeX-13B model. Instead, they are called via APIs during RoboCodeX inference to provide additional information.

E. Joint prediction of Articulated Objects

In this work, we adopt GAMMA (Yu et al., 2023b) as the base model and then fine-tune it using improved point clouds from multiple views. This is because GAMMA only uses the point cloud obtained from a single depth camera, resulting in domain gaps in our experimental setting. Except for the point cloud sources, all settings remain the same as in GAMMA. The GAMMA framework first analyzes articulated objects by segmenting them into distinct rigid parts and estimating their articulation parameters from a partial point cloud observation $P = \{\mathbf{p}_i \in \mathbb{R}^3\}_{i=1}^N$. GAMMA utilizes a PointNet++ backbone for feature extraction and has three key components: segmentation, offset, and joint parameter estimation.

1. **Segmentation Head:** It classifies each point into categories - static, revolute, or prismatic ($\hat{c}_i \in \{0, 1, 2\}$).
2. **Offset Head:** It computes offset vectors $\left(\hat{\mathbf{o}}_i \in \mathbb{R}^3\right)$, guiding points towards their part centroids.
3. **Joint Parameter Head:** It predicts joint parameters, projecting each point onto the joint axis $\left(\hat{\mathbf{v}}_i \in \mathbb{R}^3\right)$ and estimating the axis direction $\left(\hat{\mathbf{d}}_i \in \mathbb{R}^3\right)$.

The training involves minimizing a loss function, combining segmentation, offset, and joint parameter losses:

$$\mathcal{L} = \frac{1}{N}\sum_i^N \left[\mathcal{L}_c(\hat{c}_i, c_i) + \mathcal{L}_o(\hat{\mathbf{o}}_i, \mathbf{o}_i) + \mathcal{L}_v(\hat{\mathbf{v}}_i, \mathbf{v}_i) + \mathcal{L}_d(\hat{\mathbf{d}}_i, \mathbf{d}_i)\right]$$

Here, $\mathcal{L}_o(\hat{\mathbf{o}}_i, \mathbf{o}_i)$ is the offset loss, calculated as:

$$\mathcal{L}_o(\hat{\mathbf{o}}_i, \mathbf{o}_i) = \|\hat{\mathbf{o}}_i - \mathbf{o}_i\| - \left(\frac{\mathbf{o}_i}{\|\mathbf{o}_i\|_2} \cdot \frac{\hat{\mathbf{o}}_i}{\|\hat{\mathbf{o}}_i\|_2}\right)$$

After training, we freeze parameters for feature inference. We perform clustering using DBSCAN, grouping points into part clusters based on the predicted semantics and axis-aware shifts $(\mathbf{p}_i + \hat{\mathbf{o}}_i)$ and $(\mathbf{p}_i + \hat{\mathbf{v}}_i)$. Each segmented part assists in voting for joint parameters.

The final performance on unseen testing set is shown in Table 5.

Table 5. Evaluation on articulated objects joint prediction.

Objects	AP 75	Axis Direct Error	Axis Trans Error	Joint Type Acc (%)
Cabinet	83.13	6.57	10.52	97.5
Microwave	95.72	2.56	3.71	100
Refrigerator	89.75	8.92	5.36	100

F. Grasp Pose Prediction

F.1. Grasp Pose Proposal Generation

In the realm of robotic grasping, the effective generation of grasp poses is pivotal for successful object manipulation. This paper delineates two distinct models for Grasp Pose

Proposal Generation, allowing an agent to choose based on preferences inferred by the advanced algorithmic framework, RoboCodeX.

The first model, Anygrasp, is documented in Fang et al. (2023) (Fang et al., 2023). Anygrasp is a sophisticated pre-trained model adept at generating grasp poses. It utilizes a single RGB image and a corresponding point cloud to propose collision-free grasps, specifically tailored for a parallel jaw gripper. The model's efficacy lies in its ability to interpret complex scenes and propose viable grasping solutions.

The second model, herein referred to as the "central_lift" method, takes a different approach by generating top-down grasp poses. This method simplifies the grasp generation process, focusing on the central lifting points of objects, which often align with their center of mass.

Further, to refine the grasp selection process, three specific preferences are incorporated into the function "parse_adaptive_shape_grasp_pose()". It is crucial to select these preferences judiciously, as they vary significantly depending on the task requirements and the physical characteristics of the targeted objects. The preferences are as follows:

- **Preferred Position:** An Optional parameter, indicating the preferred position for the gripper tip point.
- **Preferred Approach Direction:** An Optional parameter, denoting the preferred approach direction for the gripper.
- **Preferred Plane Normal:** An Optional parameter, representing the preferred normal direction of the gripper's plane.

F.2. Grasp Execution

Upon the determination of an optimal grasp pose, the subsequent phase involves its execution. Leveraging the insights from Dasari et al. (2023) (Dasari et al., 2023), a simplified pre-grasp strategy is employed. Let $\overrightarrow{p}$ symbolize the identified grasp point, and $\overrightarrow{a}$ represent the approach vector as suggested by the grasping model. The trajectory followed by the robot gripper is mathematically expressed as:

$$\langle \overrightarrow{p} - 0.1\overrightarrow{a}, ; \overrightarrow{p} - 0.08\overrightarrow{a}, ; \overrightarrow{p} \rangle$$

This trajectory indicates an approach towards the target object from a pre-defined position, progressively reducing motion increments to ensure precision and stability. The gradual deceleration is a critical aspect of the process, especially for handling lightweight or delicate objects that are susceptible to displacement or damage through abrupt movements.

The final phase of the grasp involves a closed-loop actuation of the gripper, ensuring a secure yet careful grip on the object. This process is meticulously controlled to avoid exerting excessive force, thereby preventing potential damage to the object being manipulated.

G. Ablation Studies on Visual Reasoning Capabilities

To validate the contribution of the model's visual reasoning capabilities to navigation tasks, we conducted ablation studies as follows:

- We examined the effectiveness of the vision adapter, a module within the vision model of RoboCodeX, which is specifically designed to enhance visual perception capabilities.
- We assessed the effectiveness of the visual caption in the current scene for navigation decision-making. To do this, we removed the visual caption from the scene and input only the frontier information, similar to the baseline. This allowed the language model to make decisions based solely on the frontier information.

The results are presented in the table below:

Model	HMM3D-SR	HMM3D-SPL	HSSD-SR	HSSD-SPL
RoboCodeX	40	24.2	40.3	22.0
w/o Vision Adapter	39.2	23.5	40.1	21.5
w/o Visual Caption	38.0	22.5	39.6	21.0

Table 6. Impact of vision adapter and visual caption on navigation performance.

The results indicate that visual comprehension abilities significantly benefit navigation performance. We further analyzed the performance of RoboCodeX and RoboCodeX without the visual adapter on the general VQA (Visual Question Answering) benchmark, the LLaVA bench, and observed a positive correlation between VQA performance on the general benchmark and navigation performance. This suggests that enhanced scene understanding capabilities are greatly beneficial for navigation.

Model	LLaVA Bench
RoboCodeX	71.5
w/o Visual Adapter	67.4

Table 7. Impact of the visual adapter on general VQA performance.

These findings underscore the importance of integrating visual reasoning into models designed for navigation tasks, as it leads to improved outcomes in both general visual question answering and specific navigation challenges.

H. Real world experiments

To assess the cross-platform generalization capability of our RoboCodeX framework, we conducted experiments involving two distinct robot arms: the Franka Emika Panda robot arm and the UR5 robot arm, all without any task-specific fine-tuning tailored to each platform. Despite the inherent differences in environments and object sets between the two robotic systems, RoboCodeX delivered impressive results on both platforms by simply modifying the robot configuration file. As demonstrated in Figure 12, the UR5 robot arm exhibited remarkable precision in tasks involving fruit and cosmetic manipulations, while the Franka Emika Panda robot arm adeptly positioned a toy on a car as per the given instructions. For each task, we performed 10 random trials with randomized initial object positions. The adaptability of our system across these robot arms, without the need for platform-specific training, underscores its versatility for multi-robot deployment. These zero-shot transfer experiments vividly showcase RoboCodeX's ability to generalize control policies effectively to new testbeds with minimal reconfiguration, making it a valuable asset for scalable real-world robot learning. See more quantitive results in Appendix H.

I. Introduction of the APIs and the prompts

The provided Python application programming interface (API) enables code generation for robot control and planning within robot operating systems. As demonstrated in Listing 2 and Listing 3, the APIs we provided contain modules for operating system interactions via ROS, robot perception utilities for sensing environments, and motion planning capabilities. Key perception functions determine object positions, bounding boxes, grasp poses, detected objects, and plane normals. Manipulation functions handle attaching, detaching, and moving objects. For motion, the API offers path planning and execution, including generating paths around joints and following predefined trajectories. Gripper poses and grasps can be controlled explicitly. As shown in Listing 4, we also design prompts to guide large foundation models in adaptively generating robotic grasp poses. Our method leverages preferred positions, approach directions, and plane normals to create task-oriented gripper alignments and orientations. By using preferences rather than hard-coded values, our approach enables versatile grasping behavior tailored to specific tasks and objects. The rationale behind these preferences is crucial - they select appropriate grasps aligned and oriented to the task and target. Depending on complexity, additional helper functions may be required, definable in external scripts to ensure completion. For safety and effectiveness, we establish boundaries within the robot's workspace. We also advocate avoiding hard-coded values in prompts where possible, instead using APIs and tools to dynamically adapt values to varying tasks and objects. This ensures responsiveness. Finally, we strictly emphasize that only Python code and comments starting with '#' can be generated.

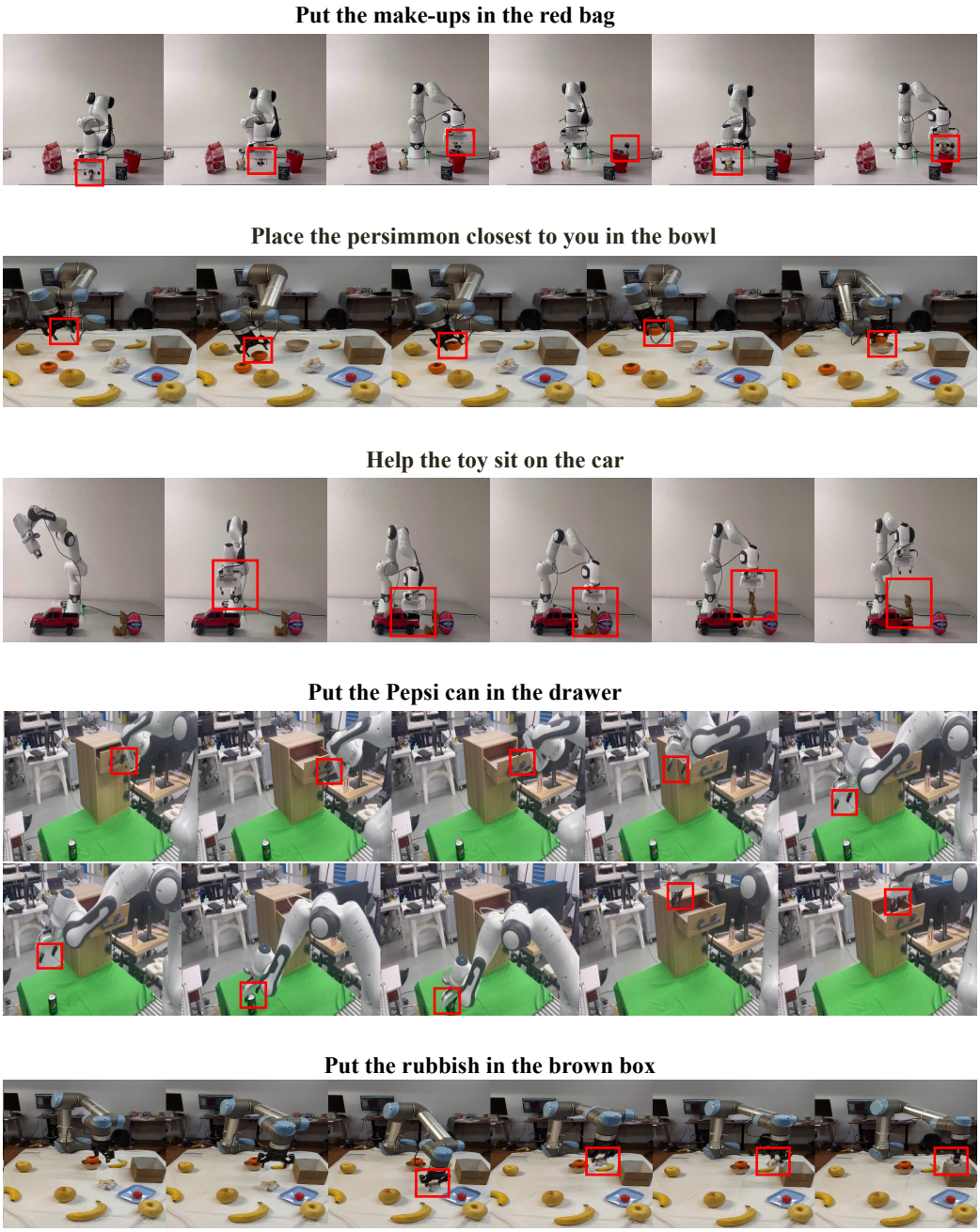

Figure 12. Real world experiments on Franka Emika Panda robot arm and UR5 robot arm. Success rates for tasks: putting make-ups in the red bag (80%), placing the persimmon closest to you in the bowl (100%), helping the toy sit on the car (90%), putting the Pepsi can in the drawer (80%), and putting the rubbish in the brown box (100%).

Listing 2. Python API for Code Generation (part one)

```
#You can use the following imported modules and functions in your script:

# Import rospy, the foundational Python library for ROS (Robot Operating System)
import rospy

# Import move_group, the class handle for moving the robot arm in MoveIt!
import move_group

# Import various ROS message types for handling robot pose and orientation
from geometry_msgs.msg import PoseStamped, Pose, Point, Quaternion

# Import utility functions for perception
from perception_utils import (
    get_object_center_position,  # Returns the position of an object in the world frame.
        Returns: position: np.array [x,y,z]
    get_object_pose,              # Returns the pose of an object in the world frame.
        Returns: pose: Pose
    get_3d_bbox,                 # Returns the 3D bounding box of an object in the world
        frame. Args: object_name: str. Returns: bbox: np.array [x_min, y_min, z_min, x_max
        , y_max, z_max]
    get_obj_name_list,           # Returns a list of names of objects present in the scene
    parse_adaptive_shape_grasp_pose, # Parse adaptive grasp pose for objects. Args:
        object_name: str, preferred_position: Optional(np.ndarray); preferred gripper tip
        point position; preferred_approach_direction: Optional(np.ndarray), preferred
        gripper approach direction; preferred_plane_normal: Optional(np.ndarray),
        preferred gripper plane normal direction. Returns: grasp_pose: Pose
    parse_central_lift_grasp_pose, # This method involves a vertical lifting action. The
        gripper closes at the center of the object and is not suitable for elongated
        objects and is not suitable for the objects with openings, as the gripper's width
        is really small. It is optimal for handling spherical and cuboid objects without
        any opening that are not ideal for off-center grasping. Args: object_name: str,
        description: Optional(str) in ['top', 'center'], Returns: grasp_pose: Pose
    parse_place_pose,            # Predict the place pose for an object relative to a
        receptacle. Args: object_name: str, receptacle_name: Optional(str), position:
        Optional(np.array) [x,y,z], . Returns: place_pose: Pose
    detect_objects,              # Detect and update task-relevant objects' status in the
        environment. Call this function before interaction with environment objects. Args:
         object_list: Optional(List[str]), objects to detect.
    get_object_joint_info,       # Get the joint info of an object closest to a given
        position. Args: obj_name: str, name of the object; position: np.ndarray, select
        the joint closest to this position; type: str, allowed type of the joint, choice
        in ['any', 'revolute', 'prismatic']. Returns: joint_info: dict, joint info of the
        object. {'joint_position':[x,y,z],'joint_axis':[rx,ry,rz],'type':str}
    get_plane_normal,            # Get the plane normal of an object closest to the given
        position. Args: obj_name: name of the object position: np.ndarray, select the
        plane closest to this position. Returns: np.ndarray, normal vector [x,y,z]
)
```

Listing 3. Python API for Code Generation (part two)

```
# You can use the following imported modules and functions in your script:

# Import utility functions for robot motion planning and execution
from motion_utils import (
    attach_object,  # Attaches an object to the robot gripper in the planning space. Call
        this function right after closing the gripper. Args: object_id: str.
    detach_object,   # Detaches an object from the robot gripper in the planning space.
        Call this function right after opening the gripper. Args: object_id: str.
    open_gripper,    # Open the gripper. No args.
    close_gripper,   # Close the gripper. No args.
    move_to_pose,    # Move the gripper to pose. Args: pose: Pose
    move_in_direction, # Move the gripper in the given direction in a straight line by
        certain distance. Note that you can use the surface normal and the joint axis.
        Args: axis: np.array, the move direction vector ; distance: float.
    generate_arc_path_around_joint, # Generate a rotational gripper path of poses around
        the revolute joint. Args: current_pose: Pose, current pose of the gripper;
        joint_axis: np.array, the joint axis of the revolute joint; joint_position: np.
        array, the joint position of the revolute joint; n: int, number of waypoints;
        angle: float, angle of rotation in degree. Returns: path: List[Pose]
    follow_path,     # Move the gripper to follow a path of poses. Args: path: List[Pose]
    get_gripper_pose, # Get the gripper pose. No args. Returns: pose: Pose
    grasp,           # Executes a grasp motion at the grasp_pose. Args: grasp_pose: Pose
)
You are encouraged to use above APIs to complete the task.
```

Listing 4. Other Prompts for Code Generation

```
# There are three preferences for function parse_adaptive_shape_grasp_pose().
# Note that you need to choose the right preferences for different tasks and objects.
# preferred_position: Optional(np.ndarray), preferred gripper tip point position.
# preferred_approach_direction: Optional(np.ndarray), preferred gripper approach direction
    .
# preferred_plane_normal: Optional(np.ndarray), preferred gripper plane normal direction.
# This preference selects the grasp pose so that the gripper is parallel to the plane
    defined by the normal.
# It also means the gripper grasping at an axis with perpendicular pose to the axis.
# Note that you may always need to create your arbitrary functions to help you complete
    the task, which will be defined by external scripts.
# The robot working space on table top is in range [-0.5, 0.2] in x-axis and [-0.5, 0.5]
    in y-axis. The height of the table top is 1.05.
# Note that you are a strict coder, you can not hard code a predefined value, but you need
     to use api and tools to detect the value.
# Please pay attention to the description and specific requirements of the target object
    in the task. You may need to write some founctions by yourself to determine whether
    the requirements are met.
# Your generated content should only contain comments starting with '#' and python code!
```