Octopus: Embodied Vision-Language Programmer from Environmental Feedback

Jingkang Yang*,1, Yuhao Dong*,2,3, Shuai Liu*,2,4, Bo Li*,1, Ziyue Wang†,1, Haoran Tan†,4, Chencheng Jiang†,5, Jiamu Kang†,3, Yuanhan Zhang1, Kaiyang Zhou6, and and Ziwei Liu1,✉

[1] S-Lab, Nanyang Technological University [2] Shanghai AI Laboratory
[3] Tsinghua University [4] BUPT [5] XJTU [6] Hong Kong Baptist University
{jingkang001, ziwei.liu}@ntu.edu.sg

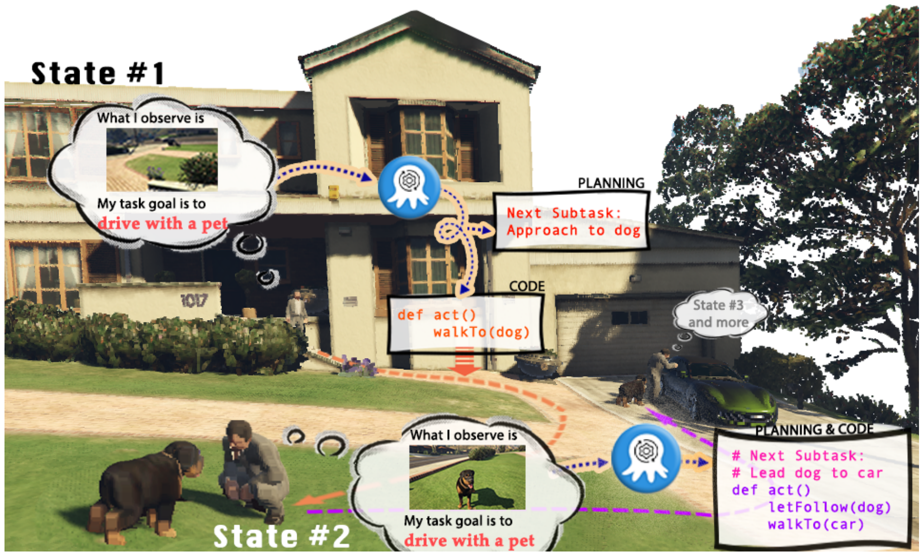


Fig. 1: Illustration of our vision-language programmer, Octopus, complete a task in GTA environment. Given a task in the form of natural language, Octopus relies on its egocentric vision to generate plans and the corresponding executable code.

Abstract. Large vision-language models (VLMs) have achieved substantial progress in multimodal perception and reasoning. When integrated into an embodied agent, existing embodied VLM works either output detailed action sequences at the manipulation level or only provide plans at an abstract level, leaving a gap between high-level planning and real-world manipulation. To bridge this gap, we introduce **Octopus**, *an embodied vision-language programmer* that uses executable code generation as a medium to connect planning and manipulation. Octopus is designed to **1)** proficiently comprehend an agent's visual and textual task objectives, **2)** formulate intricate action sequences, and **3)** generate executable code. To facilitate Octopus model development, we introduce **OctoVerse**: a suite of environments tailored for benchmarking

* Equal contribution, †Equal engineering contribution, ✉ Corresponding author.

vision-based code generators on a wide spectrum of tasks, ranging from mundane daily chores in simulators to sophisticated interactions in complex video games such as Grand Theft Auto (GTA) and Minecraft. To train Octopus, we leverage GPT-4 to control an explorative agent that generates training data, i.e., action blueprints and corresponding executable code. We also collect feedback that enables an enhanced training scheme called **Reinforcement Learning with Environmental Feedback (RLEF)**. Through a series of experiments, we demonstrate Octopus's functionality and present compelling results, showing that the proposed RLEF refines the agent's decision-making. By open-sourcing our simulation environments, dataset, and model architecture, we aspire to ignite further innovation and foster collaborative applications within the broader embodied AI community. The project page is available at `https://choiszt.github.io/Octopus/`.

1 Introduction

The rise of large language models (LLMs) [11,15,42,46,54] led to a surge in vision-language models (VLMs) [4,5,31,33,35–37], enabling tasks such as image/video-based descriptions [33], reasoning [14,38,60], and conversations [16,31]. In the realm of embodied AI, notable efforts [10,18,33] have trained agents to process visual input and relay motor control commands.

Another approach to interacting with the environment focuses on task execution through code invocations, mirroring the human System-I stimulation [19,29] (automatic, intuitive actions) with predefined code, and leaving the System-II processes [19,29] (planning and reasoning) for large models. For example, referring to Fig. 1, planning a car ride with a pet might entail a subconscious checklist (e.g., `getOutOf()` the house, `open()` the car door), each action could be implemented using specific techniques [8,24] such as imitation learning [22,28]. This programmatic paradigm has been, although not in vision, leveraged by works [25, 50, 52, 53] using LLMs to craft programs and trigger APIs. Game-centric models like Voyager [55] have similarly employed GPT for function calls within game engines, though they often parse data directly from their environments.

However, when incorporating visual perception, the programming paradigms are largely unexplored. Primary initiatives [48,58] can only output plans, which anchor their strategies only in initial environmental states or employ dynamic scene graphs for LLM inputs, respectively. Despite their innovations, the over-reliance on pre-trained vision models to convert vision content into language can occasionally hinder the LLM's planning performance. The conversion from plans into real-world actions is still missing. While EmbodiedGPT [41] addresses the problem by integrating vision-language modeling for planning and then transitioning to manipulation using policy mapping, the capability of embodied vision-language models to generate executable programs remains largely uncharted.

Our exploration aims to bridge this gap. An embodied vision-language programmer should integrate visual perspective with textual objectives to devise action plans and executable code (Fig. 1). However, existing simulator environments often lack the carefully designed functions necessary to support such models effectively. These functions should balance usefulness and complexity to avoid hindering the development of genuine embodied vision-language programmers. For example, the `explore_until()` function in Minecraft, which can lead the player directly to specific blocks without relying on vision information, may not be suitable for training these models.

To meet the requirement, we carefully design and develop OctoVerse, a suite of environments consisting of diverse simulators, including (i) OctoGibson, built upon the photorealistic OmniGibson [32], (ii) OctoMC, developed on the infinitely creative, pixel-style Minecraft platform [2], and (iii) OctoGTA, adapted from the highly interactive and immersive Grand Theft Auto V (GTA-V) [1]. These environments enable the training and benchmarking of our embodied vision-language programming model in a wide range of scenarios, from daily household tasks to complex urban navigation and open-world exploration, while the function calls are tailored to be vision-dependent.

Using the OctoVerse environment, we train Octopus by leveraging GPT-4 to collect data. We provide GPT-4 with system messages, environmental cues, and objectives, enabling it to formulate action strategies and code. Simultaneously, the agent captures visual perspectives, forming the image-code pair for Octopus training. During data collection, the agent receives simulator feedback, distinguishing successful moves from unsuccessful ones. We incorporate this feedback using Reinforcement Learning with Environmental Feedback (RLEF) and fine-tune Octopus using Proximal Policy Optimization (PPO) [51]. Empirically, Octopus demonstrates strong adaptability in various scenarios, outperforming existing models in task planning, code generation, and execution. The integration of RLEF further enhances Octopus's performance, showcasing the effectiveness of this training approach. In sum, our key contributions include:

- **A Novel Vision-Language Programming Benchmark:** Three diverse embodied environments with designed tasks: (i) OctoGibson, which is developed upon OmniGibson [32], (ii) OctoMC that developed on Minecraft [2], and (iii) OctoGTA, which is adapted from GTA-V [1].
- **A New Vision-Language Programming Model:** An embodied vision-language planner and programmer trained with Reinforcement Learning with Environmental Feedback (RLEF), demonstrates compelling results.
- **Insights on Vision-Language Programming:** We extensively explore Octopus and share useful insights facilitating future research on visual planning and programming.

2 Related Work

2.1 Embodied AI Simulators

Embodied AI has advanced significantly with the development of diverse simulation environments, enabling research tasks such as visual exploration [47],

Table 1: Related Work for OctoVerse - Overview of Embodied AI Environments. We select three environments into OctoVerse and carefully design executable tasks and vision-dependent function calls (VC), in comparison to undesigned standard function calls (C).

Simulator	**Kinematics**	**Continuous Extended States**	**Flexible Materials**	**Deformable Bodies**	**Realistic Action Execution**	**Game- or World-Based**	**Formulated Tasks**	**Function Call Type**
OpenAIGym [9]	✓	×	×	×	✓	G	×	C
Matterport3D [12]	×	×	×	×	×	W	×	×
AI2THOR [30]	✓	×	×	×	×	G	×	C
VirtualHome [44]	×	×	×	×	×	G	×	×
House3D [57]	×	×	×	×	×	W	×	×
Habitat 1.0 [49]	✓	×	×	×	✓	W	×	C
Robosuite [66]	✓	×	×	×	✓	W	×	C
RFUniverse [21]	✓	×	✓	✓	✓	W	×	C
Minecraft [2]	✓	×	✓	×	✓	G	×	C
OctoMC	✓	×	✓	×	✓	G	✓	VC
GTA [1]	✓	✓	✓	✓	✓	G	×	C
OctoGTA	✓	✓	✓	✓	✓	G	✓	VC
OmniGibson [32]	✓	✓	✓	✓	✓	W	×	C
OctoGibson	✓	✓	✓	✓	✓	W	✓	VC

navigation [56], and question-answering [17]. Several simulators, including AI2-THOR [30], VirtualHome [44], Habitat-Sim [49], SAPIEN [59], and Omnigibson [32], provide realistic representations of the world for investigating embodied AI challenges. OmniGibson [32] stands out for its high-fidelity simulation of diverse indoor and outdoor environments. OctoGibson environment further enhances OmniGibson with carefully designed function calls and formulated tasks, making it well-suited for vision-language programming.

Game-related simulators like Arade [7], CHALET [61], and VRKitchen [23] also contribute significantly to embodied AI. Minecraft [2] has gained attention in reinforcement learning and game agents [6,20,39,55,63,64] but lacks the necessary structure for vision-language programming. OctoMC addresses this by providing designed function calls and formulated tasks. In contrast to Minecraft's voxel-based representations that limit transferability to real-world environments, GTA-V [1] offers a highly realistic environment. In this work, we introduce OctoGTA as a new setting, leveraging GTA-V's rich, open-world environment with incorporated tasks and function calls, extending this platform for embodied AI study.

2.2 Embodied AI with Large Models

The recent wave of research focuses on merging LLMs with embodied AI tasks [11, 42,46,54]. For instance, VoxPoser addresses robotic manipulation problems through unsupervised methods [27]. A group of projects, namely SayCan [3], Palm-E [18], RT-2 [10], and EmbodiedGPT [41], effectively integrate visual or linguistic cues with robot manipulation data. Outside the domain of robotic manipulation, initiatives like Voyager [55] and Smallville [43] harness the capabilities of GPT to interface with game functions, relying on preset functions to manage intricate manipulations. In a parallel vein, VisProg [25] leverages GPT-3 language prompts to craft Python programs, opening the door to a multitude of fascinating applications. While the proposed Octopus model also formulates plans

Table 2: Related Work for Octopus - Overviewing Embodied AI Models. The proposed Octopus distinguishes itself from other models as a unified vision-language model for both plan and code generation.

Models	**Release Date**	**Supported Environment**	**Vision Model**	**Code Generator**	**Action w/ Feedback**	**LLM Training Enabled**
Text2Motion [34]	Mar. 2023	Sim	×	✓	✓	×
Instruct2Act [26]	May 2023	Sim	×	✓	×	×
Lang2Rewards [62]	Jun. 2023	Sim	×	✓	✓	×
VoxPoser [27]	Jul. 2023	Sim	✓	×	×	×
SayCan [3]	Apr. 2022	Real	✓	×	✓	×
PALM-E [18]	Mar. 2023	Sim, Real	✓	×	✓	✓
RT-2 [10]	Jul. 2023	Real	✓	×	✓	✓
SayPlan [48]	Jun. 2023	Real	×	×	✓	×
EmbodiedGPT [41]	May 2023	Sim	✓	×	✓	✓
TaPA [58]	Jul. 2023	Sim	×	×	×	✓
Voyager [55]	May 2023	Game	×	✓	✓	×
Steve-Eye [64]	Dec 2023	Game	✓	×	✓	✓
RoboScript [13]	Feb 2024	Sim, Real	×	✓	✓	×
Octopus	-	Sim, Game	✓	✓	✓	✓

and code, its distinguishing feature is the seamless integration of visual input in program and code generation. This also stands in contrast to other embodied planners like TAPA [58] and SayPlan [48], which deploy separate vision modules to translate visual data into linguistic inputs for LLMs. Octopus excels as a cohesive vision-language model, delivering not just plans but also code.

More discussion on additional related works (Vision Language Model, and Feedback in Large Language Models) is included in the supplementary materials.

3 The OctoVerse Environment

In this section, we introduce three simulator environments designed to train and evaluate the Octopus model. For each environment, we will describe their overall information, the special design considerations that ensure the tasks are well-formulated and the callable functions are vision-dependent.

OctoGibson We built the environment on the foundation of OmniGibson [32], an existing simulation framework that supports 1,000 daily activities across 50 scenes, featuring over 5,000 meticulously annotated objects. We incorporated 16 functions that the robot can execute, such as `moveBot()` and `easyGrasp()`. Within this environment, we meticulously crafted 476 tasks[1], each with a well-defined initial state and a definitive termination state, allowing for a straightforward assessment of task completion. Among these tasks, 367 are routine tasks—simple and direct actions like “place a glass in a trash can,” marked as `Follow`. Conversely, the remaining 109 are reasoning tasks that necessitate deeper comprehension. An example is “buy a chocolate,” where the agent needs

[1] The full list of tasks and their categories are listed in the supplementary material.

to know to pick a chocolate bar from the shelf and then place it, along with money, on the checkout counter, denoted as `Reason` tasks.

To ensure our tasks are vision-aware, we deliberately constrain the usage of certain functions, such as `moveBot(object)`, which moves the agent in front of the given `object`. To avoid making the task too easy and vision-agnostic, we limit the given parameter to a predefined set of large objects, such as tables and cupboards, rather than small items like cups and glasses. In this case, if the robot wants to pick up a cup, it needs to recognize whether the cup is on the table or in the cupboard. A simple `moveBot(object)` call with an inappropriate parameter would cause a runtime error. The full list of the functions is in the Appendix.

OctoMC The OctoMC environment is built on Minecraft [2], a popular platform for reinforcement learning and game agents. We integrated 6 functional actions and crafted 40 tasks[2], each designed to facilitate comprehensive observations and executions by the agent. These tasks are distributed across 10 different biomes, including indoor, outdoor, and underground settings, under varying weather conditions.

However, existing Minecraft environments often lack vision-formulated tasks and the necessary structure for vision-language programming. For example, in Voyager [55], the `exploreUntil()` function allows the player to navigate directly to specific blocks without relying on vision information. This function works by randomly exploring the environment within a certain range until the desired object is found, at which point it returns a true value, enabling the agent to interact with the located object, such as gold blocks or trees. While effective, this approach is entirely automated and does not utilize visual information, making it unsuitable for our vision-based objectives.

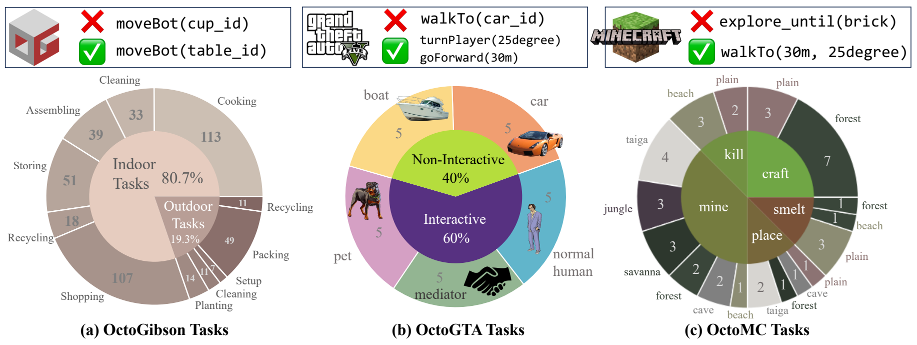
To address this limitation, we crafted a vision-dependent exploration function, `teleport(yaw, distance)`, which operates within the robot's perceptual range. This function ensures that the agent's operations are vision-dependent and require active perception and navigation within the environment. For instance, to locate a tree block, the agent must actively navigate towards the direction of the forest, relying on visual cues to guide its exploration.

OctoGTA The OctoGTA environment is built on GTA-V [1] with the help of the active GTA modding community. We integrated 19 functions and methodically crafted 25 tasks[3], such as "help NPC drive their boat back to shore" and "mediate a fight between two NPCs," spanning across 5 groups (shown in Fig. 2 (b)). Each task is assigned to 5 different locations within the game world.

We have implemented a set of functions that enable the agent to interact with the game world in a visually-aware manner. Similar to the design in Minecraft, we get rid of functions like `walkTo(location)` that might trivialize the task of reaching a particular building or landmark. Instead, we provide functions such

[2] Detailed tasks are listed in the supplementary material.

[3] We meticulously designed tasks to be friendly, ensuring they exclude any inappropriate or violent behaviors.



```
moveBot(cup_id)
moveBot(table_id)
walkTo(car_id)
turnPlayer(25degree)
goForward(30m)
explore_until(brick)
walkTo(30m, 25degree)
```

Fig. 2: The Statistics of the OctoVerse Environment with Function Designs.

as `goForward(distance)` and `turnPlayer(degree)` (Fig. 2 (b)). For tasks like mediating a fight between two NPCs, the essential function `stopFight()` only works when the player is within 5 meters of the fighting NPCs. These design choices ensure that the agent's operations are vision-dependent and require active perception and navigation within the environment.

4 Octopus: The Embodied Vision-Language Programmer

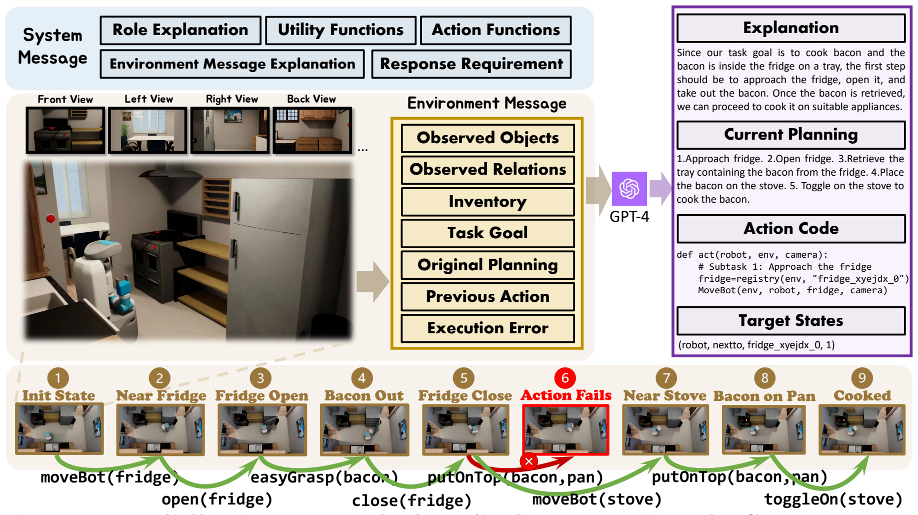
In this section, we present the procedure for training Octopus. Starting from collecting training data within the OctoVerse environment, Octopus builds upon a VLM architecture of Otter [31] and includes specialized RLEF modules to handle vision-language programming tasks. Fig. 4 illustrates the entire Octopus training pipeline.

4.1 Training Data Collection

We use the automatic training data collection pipeline described here for OctoGibson and OctoMC, with the latter using customized prompts inspired by Voyager [55]. For OctoGTA, we rely on human labor to hand-craft the training dataset due to the difficulty of obtaining textual environment messages in the GTA environment. In the following parts, we use OctoGibson as the primary example to illustrate the data collection pipeline. Note that the primary task in organizing training data is to form a succinct pairing: "vision input + current/historical states $\rightarrow$ next step plan + executable code".

Environment Info Collection As shown in Fig. 4 (a) and Fig. 3, we format an **environment message** for each state, encompassing attributes like `Observed Objects`, `Observed Relations`, `Inventory`, and more. Specifically, the simulator can provide us with an exact scene graph at each state, shaping the content for the first two parts. The inventory info is also accessible. The task, e.g., "cooking bacon" in Fig. 3, is represented by the `Task Goal`.

Automation with GPT-4 After preparing the environment message, we crafted a structured **system message** to ensure that the robot not only understands its input but also maintains a consistent output format. A detailed



Since our task goal is to cook bacon and the bacon is inside the fridge on a tray, the first step should be to approach the fridge, open it, and take out the bacon. Once the bacon is retrieved, we can proceed to cook it on suitable appliances.

1.Approach fridge. 2.Open fridge. 3.Retrieve the tray containing the bacon from the fridge. 4.Place the bacon on the stove. 5. Toggle on the stove to cook the bacon.

```
def act(robot, env, camera):
    # Subtask 1: Approach the fridge
    fridge=registry(env, "fridge_xyejdx_0")
    MoveBot(env, robot, fridge, camera)
```

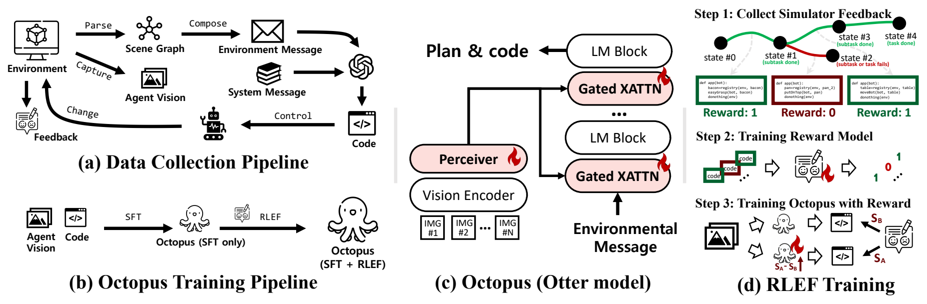
(robot, nextto, fridge_xyejdx_0, 1)

Fig. 3: Data Collection Example for "Cook a Bacon" Task. GPT-4 perceives the environment through the `environmental message` and produces anticipated plans and code following the detailed `system message`. This code is subsequently executed in the simulator, directing the agent to the subsequent state. For each state, we gather the environmental message, wherein `observed objects` and `relations` are substituted by egocentric images to serve as the training input. The response from GPT-4 acts as the training output. Environmental feedback, specifically the determination of whether each target state is met, is documented for RLEF training.

examination of this prompt can be found in the appendix. Experiments have shown that a well-articulated prompt enables GPT-4 to effectively generate executable code. It is important to note that the combined length of the system and environment messages can be extremely long, which may cause standard GPT-4 8K models to struggle with producing meaningful outputs. To address this issue, we employ the more robust GPT-4 32K model. As illustrated in Fig. 3, when GPT-4 receives a consistent system and environment message, it generates comprehensive outputs that include current scenario analysis, planning, and actionable code, supporting the training process in Section 4.3.

Error Management Notably, GPT-4 collects training data under the main task of guiding the agent to complete tasks. However, GPT-4 is not infallible. Errors can manifest in multiple ways, ranging from syntax errors to physical challenges in the simulator. For instance, in Fig. 3, between states #5 and #6, the action failed due to the long distance between the agent (holding bacon) and the pan. Such setbacks reset the task to its previous state. If a main task remains incomplete after 10 steps, it is deemed unsuccessful, and we terminate this task for budget concerns. However, all data pairs without syntax errors, regardless of the task's completion status, are valuable for refining instructions and improving the model's performance.

Environmental Feedback GPT-4's continual trial-and-error approach while guiding the agent toward task completion serves a dual purpose: collecting vision-



```
def app(bot):
    bacon=registry(env, bacon)
    easyGrasp(bot, bacon)
    donothing(env)
```

```
def app(bot):
    pan=registry(env, pan_2)
    putOnTop(bot, pan)
    donothing(env)
```

```
def app(bot):
    table=registry(env, table)
    moveBot(bot, table)
    donothing(env)
```

Fig. 4: How to train Octopus: data collection and training pipeline.

output pairs and generating a rich set of feedback data. The automatic annotation of this feedback focuses on two levels: step-level and task-level judgments. **Step-level judgment** assesses the alignment of post-execution states with their target states. For instance, in Fig. 3, steps color-coded in green lead to positive feedback. One can visualize the action sequence for task completion as a tree, where each node indicates a step (subtask), encapsulating an action code. Accompanying each step is a binary value that denotes success or failure, giving preference to the successful branch over its counterpart. **Task-level judgment**, on the other hand, gauges the successful execution of the overall task. If the task is not completed as intended, every state within that task is labeled as negative, regardless of the status of the subtasks. This collated feedback data serves as a foundation for our Reinforcement Learning with Environmental Feedback (RLEF) methodology, which we discuss in greater detail in Section 4.4.

4.2 Model Architecture

The Octopus architecture (shown in Fig. 4 (c)) is heavily inspired by the Otter model [31], integrates the **MPT-7B Language Decoder** [40] and **CLIP VIT-L/14 Vision Encoder** [45]. Adopting design principles from Flamingo [4], Octopus employs the **Perceiver Resampler** and **Cross-Gated Attention modules** to enhance vision-language synergy. This architecture enables Octopus to excel in tasks requiring understanding of both visual and textual data. The Octopus is also compatible with other VLMs such as LLaVA [37].

4.3 SFT: Supervised Finetuning with Instructions

We train the Octopus model on our collected dataset from OctoVerse $\mathcal{D}_{\mathrm{E}} = \{(\mathbf{X}_v, \mathbf{T}_i, \mathbf{T}_r)\}$ using token-level supervised fine-tuning (SFT) [42, 54]. The Perceiver Resampler transforms images $\mathbf{X}_v$ into visual tokens that condition subsequent layers via Cross-Gated Attention modules. The training objective is next-token prediction, modeling the likelihood of a targeted response $\mathbf{T}_r$ as:

$$p(\mathbf{T}_r \mid \mathbf{T}_i, \mathbf{X}_v) = \prod_{l=1}^{L} p(t_l \mid \mathbf{X}_v, \mathbf{T}_i, \mathbf{T}_{r,<l}). \tag{1}$$

Note that $\mathbf{T}_i$ denotes the instruction tokens and $\mathbf{T}_{r,<l}$ denotes the response tokens before the current predicted token t_l. During inference, tokens are converted into natural language via the language decoder's text tokenizer.

Visual observations $\mathbf{X}_v = \{x_F^0, \ldots, x_F^7, x_B^0, x_B^1\}$ consist of 8 first-person view (FPV) images and two bird's-eye view (BEV) images for OctoGibson and OctoGTA. OctoMC only takes 4 FPV images. The FPV captures the agent's direct observations, while the BEV provides a holistic understanding of the environment. The eight FPV images are captured every 45 degrees, ensuring a complete 360-degree perspective.

4.4 RLEF: Reinforcement Learning with Environmental Feedback

In OctoVerse, task progression can be visualized as a tree (Fig. 4 (d)), where each node represents a sub-task with a binary value indicating success (1) or failure (0). If a node (or sub-task) has a value of 1, it is a step in the correct direction toward our end goal.

Tree-based Task Representation According to the environmental feedback part in Sec. 4.1, environmental reward datasets $\mathcal{D}_{\mathtt{R}} = (\mathbf{X}_v^*, \mathbf{T}_i^*, \mathbf{T}_r^i, \mathbf{T}_r^j, c)$ are organized, where $\mathbf{T}_r^i$ and $\mathbf{T}_r^j$ are responses sharing the same parent task $\mathbf{T}_i^*$, and c indicates the preferred response leading to task completion. This ensures the reward mechanism favors the successfully executed branch. Note that even if a parental node does not have multiple responses, we can still assign feedback according to the rule in Sec. 4.1.

Reward Model Configuration A single-modal CodeLLaMA-7B model with an additional value head is fine-tuned on $\mathcal{D}_{\mathtt{R}}$ as the reward model r_ϕ. This text-based model assesses state transitions ($\mathbf{T}_i^* \rightarrow \mathbf{T}_r^{i,j}$) to determine high-reward transitions, assisting the agent in task execution and completion. The rationale for using CodeLLaMA as the reward model is that evaluating rewards can be purely dependent on the textual output. Furthermore, CodeLLaMA's strong programming skills make it well-suited for assessing the quality and effectiveness of the generated code in the context of task completion.

Policy Model Development The supervised fine-tuned model serves as the initial policy model $\pi^{\mathtt{INIT}}$ with fixed parameters. A duplicate model, $\pi_\theta^{\mathtt{RL}}$, is initialized and trained using Proximal Policy Optimization (PPO) [51] to maximize response rewards. The loss function is:

$$\mathcal{L}\left(\pi_\theta^{\mathtt{RL}}\right) = -\mathbb{E}_{(\mathbf{X}_v^*, \mathbf{T}_i^*) \in \mathcal{D}_{\mathtt{R}}, \mathbf{T}_r \sim \pi^{\mathtt{RL}}} \left[r_\phi(\mathbf{T}_i^*, \mathbf{T}_r) - \beta \cdot \mathbb{D}_{\mathtt{KL}} \left(\pi_\theta^{\mathtt{RL}}(\mathbf{X}_v^*, \mathbf{T}_i^*) \parallel \pi^{\mathtt{INIT}}(\mathbf{X}_v^*, \mathbf{T}_i^*) \right) \right], \tag{2}$$

where β acts as a hyper-parameter to regulate the magnitude of the Kullback–Leibler (KL) penalty.

5 Experiments

5.1 Main Results on OctoGibson

Experimental Setup We first set up the OctoGibson to evaluate the performance of Octopus and other related models. Specifically, we are utilizing the

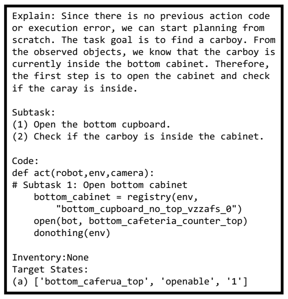
Table 3: Main Results on OctoGibson. We compare various models: standalone language models, adapted vision-language planners, and our Octopus models, across different evaluation settings. In cells displaying two values, the first represents the task completion rate across the target validation task sets, while the second assesses the conceptual accuracy of the model's planning as judged by human evaluators. GT denotes that the model input is directly parsed from the simulator, with information on objects (O) or relations (R). Octopus shows consistent advance in task completion.

Model	Vision Model	Language Model	Entire Goal Task				
			Seen Env	Unseen Env	Follow	Reason	All
GPT-4	-	-	0.42 / 0.69	0.46 / 0.67	0.49 / 0.78	0.27 / 0.40	0.43 / 0.68
GPT-4V	-	-	0.40 / 0.62	0.60 / 0.67	0.42 / 0.67	0.53 / 0.53	0.45 / 0.63
LLaMA	GT (O+R)	LLaMA2-7B	0.07 / 0.11	0.13 / 0.13	0.11 / 0.16	0.00 / 0.00	0.08 / 0.12
CodeLLaMA	GT (O+R)	CodeLLaMA-7B	0.09 / 0.20	0.20 / 0.40	0.16 / 0.31	0.00 / 0.07	0.12 / 0.25
TAPA (task-level)	~~OVD~~ GT (O)	CodeLLaMA-7B	0.09 / 0.36	0.13 / 0.33	0.11 / 0.36	0.06 / 0.33	0.10 / 0.35
TAPA (step-level)	~~OVD~~ GT (O)	CodeLLaMA-7B	**0.16** / **0.42**	0.13 / 0.27	**0.18** / 0.38	0.07 / 0.40	0.15 / 0.38
EmbodiedGPT	CLIP-ViT	MPT-7B	0.04 / 0.36	0.27 / **0.53**	0.13 / 0.38	0.00 / 0.40	0.10 / 0.40
Octopus (SFT Only)	CLIP-ViT	MPT-7B	0.11 / 0.33	0.27 / 0.47	0.16 / 0.38	0.13 / 0.33	0.15 / 0.37
Octopus (SFT + RLEF)	CLIP-ViT	MPT-7B	0.13 / 0.38	**0.33** / **0.53**	**0.18** / **0.40**	**0.20** / **0.53**	**0.18** / **0.42**

metrics of goal task completion score to check whether the task is completed in the simulator and the plan score from human evaluation. We have 60 evaluation tasks, with 45 from the seen environment, and 15 that are unseen during training. We also have 45 routine tasks and 15 require reasoning. Please note that models like Octopus might not always accurately identify specific object names as they appear in the simulator (e.g., "water_bottle_189"). To address this, we implement a post-processing step for the generated code, substituting generic object references with their exact names from the simulator with simple string similarity matching. If multiple objects, we select the one closest to the agent.

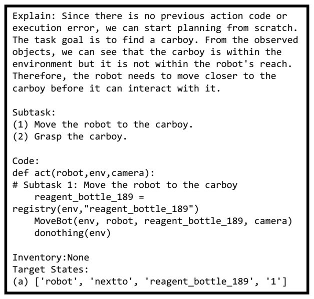
For Blind LLMs, we provide them with all the environment information in a textual format. Referring to Figure 3, we hope the Blind LLMs could perform as GPT-4 but internalize the system message. For TAPA utilizes the open-vocabulary detection (OVD) technique [65] to recognize objects within images and parse them into textual environmental messages, we still provide it with ground-truth environmental messages as an oracle setting.

CodeLLaMA Improves Coding but not Planning. The first two rows in Table 3 highlight the suboptimal task completion rate of the blind LLMs. Among them, CodeLLaMA boasts pre-training on a large programming dataset, resulting in a notable enhancement in code execution from our observation, with 92% of the written code being successfully executed compared to LLaMA's 24%. However, its prowess in planning remains limited. In contrast, the proposed Octopus MPT-7B model displays superior planning and task completion metrics while maintaining commendable coding abilities (72% of the written code can be executed). We surmise that the coding requirements within the OctoGibson environment might not be exceedingly intricate, rendering an advanced programming language model, like CodeLLaMA, less crucial, albeit beneficial. For more insight, although not shown in the table, our efforts to replace the MPT model with CodeLLaMA encountered challenges of generating non-sense outputs, suggesting that more refined code, or image-code paired data might be necessary for a successful Octopus-CodeLLaMA integration.



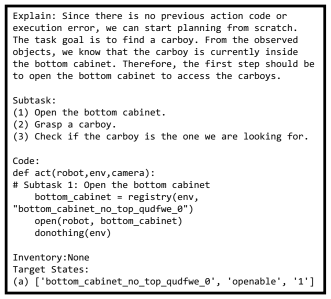
```
Explain: Since there is no previous action code
or execution error, we can start planning from
scratch. The task goal is to find a carboy. From
the observed objects, we know that the carboy is
currently inside the bottom cabinet. Therefore,
the first step is to open the cabinet and check
if the caray is inside.

Subtask:
(1) Open the bottom cupboard.
(2) Check if the carboy is inside the cabinet.

Code:
def act(robot,env,camera):
# Subtask 1: Open bottom cabinet
    bottom_cabinet = registry(env,
        "bottom_cupboard_no_top_vzzafs_0")
    open(bot, bottom_cafeteria_counter_top)
    donothing(env)

Inventory:None
Target States:
(a) ['bottom_caferua_top', 'openable', '1']
```

(a) TAPA (step) + CodeLLaMA



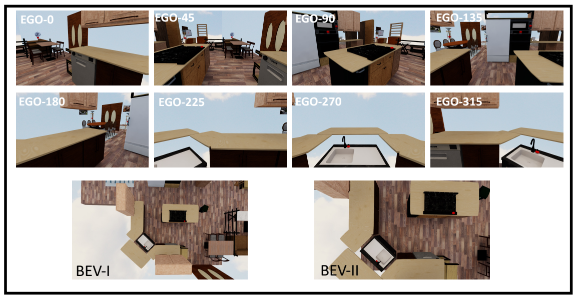
```
Explain: Since there is no previous action code or
execution error, we can start planning from scratch.
The task goal is to find a carboy. From the observed
objects, we can see that the carboy is within the
environment but it is not within the robot's reach.
Therefore, the robot needs to move closer to the
carboy before it can interact with it.

Subtask:
(1) Move the robot to the carboy.
(2) Grasp the carboy.

Code:
def act(robot,env,camera):
# Subtask 1: Move the robot to the carboy
    reagent_bottle_189 =
registry(env,"reagent_bottle_189")
    MoveBot(env, robot, reagent_bottle_189, camera)
    donothing(env)

Inventory:None
Target States:
(a) ['robot', 'nextto', 'reagent_bottle_189', '1']
```

(b) Octopus (SFT only)

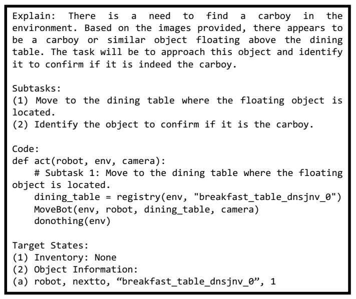


```
Explain: Since there is no previous action code or
execution error, we can start planning from scratch.
The task goal is to find a carboy. From the observed
objects, we know that the carboy is currently inside
the bottom cabinet. Therefore, the first step should be
to open the bottom cabinet to access the carboys.

Subtask:
(1) Open the bottom cabinet.
(2) Grasp a carboy.
(3) Check if the carboy is the one we are looking for.

Code:
def act(robot,env,camera):
# Subtask 1: Open the bottom cabinet
    bottom_cabinet = registry(env,
"bottom_cabinet_no_top_qudfwe_0")
    open(robot, bottom_cabinet)
    donothing(env)

Inventory:None
Target States:
(a) ['bottom_cabinet_no_top_qudfwe_0', 'openable', '1']
```

(c) Octopus (SFT + RLEF)



(d) Vision Input for Vision-Language Models (e.g., b, c, e)



```
Explain: There is a need to find a carboy in the
environment. Based on the images provided, there appears to
be a carboy or similar object floating above the dining
table. The task will be to approach this object and identify
it to confirm if it is indeed the carboy.

Subtasks:
(1) Move to the dining table where the floating object is
located.
(2) Identify the object to confirm if it is the carboy.

Code:
def act(robot, env, camera):
    # Subtask 1: Move to the dining table where the floating
object is located.
    dining_table = registry(env, "breakfast_table_dnsjnv_0")
    MoveBot(env, robot, dining_table, camera)
    donothing(env)

Target States:
(1) Inventory: None
(2) Object Information:
(a) robot, nextto, “breakfast_table_dnsjnv_0”, 1
```

(e) GPT-4V Output

Fig. 5: Qualitative Results on the task of *find a carboy* in OctoGibson environment. We show that the models shown can write executable code, but the proposed Octopus has stronger planning ability, especially after RLEF. We also explore the performance of GPT-4V on the specific task.

Blind LLMs Struggle with Extended Input Content. Our observations indicate that the step-level TAPA model, when supplied with a ground-truth object list, achieves a notable enhancement in planning. The primary distinction between it and the blind CodeLLaMA lies in the input length; the latter deals with protracted, pairwise relation content, complicating the language model's ability to extract crucial data from the environment message. This scenario highlights the inherent limitation of blind LLMs: relying on language alone to convey the entirety of environmental data can result in less informative input.

Octopus Demonstrates Superior Task Generalization. Table 3 underscores Octopus's strong performance, evidencing its consistent edge over standalone language models in task completion. Its adeptness in adapting to previously unencountered environments underlines the inherent advantages of vision-language models. A more detailed ablation analysis is provided later.

RLEF Enhances Octopus's Planning Strategy. Table 3 shows Octopus's strong reasoning capabilities after the RLEF finetuning. An example can be observed in Fig. 5(b-c), where, after refinement via RLEF, Octopus astutely navigates to the cabinet housing the carboy instead of attempting a direct yet distant capture. Quantitatively, Octopus exhibits enhanced adaptability to previously unseen reasoning tasks, reinforcing its prowess in logical task resolution. When juxtaposed with other strategies, such as the embodied queries employed by EmbodiedGPT, RLEF emerges as the more efficacious approach.

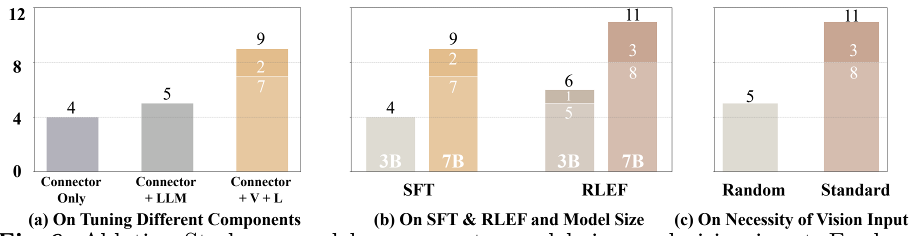


Fig. 6: Ablation Study on model components, model size, and vision input. For bars with different colors, the upper bar denotes the number of successful reasoning tasks, and the lower is routine tasks.

5.2 Ablation Study

Tunning Different Components Fig. 6 (a) demonstrates that solely adjusting the connector (marked "fire" in Fig. 4 (a)) leads to success for merely 4 out of 60 tasks. Conversely, finetuning both the connector and language decoder nudges the success rate slightly higher, with 5 tasks being accomplished.

7B *v.s.* 3B Model Size We embarked on experiments centered on model size to discern the influence of the total parameter count on the efficacy of vision-language models. As illustrated in Fig. 6 (b), downsizing the model manifests in a noticeable performance drop. The congruency of results across both the SFT and RLEF models underscores the importance of an apt model size when sculpting vision-language models.

Significance of Visual Inputs in Task Performance In our standard configuration, the vision component processes a sequence of image inputs, consisting of eight circularly captured first-person view (FPV) images, complemented by two bird's-eye view (BEV) images. With the intent to investigate the impact of visual inputs on task performance, we initiated an ablation study. In a modified setup, the sequence of these visual inputs was deliberately randomized, aiming to attenuate the strength of the visual signals. As illustrated in Fig. 6 (c), this intentional disruption in visual input consistency led to a pronounced decline in task performance. This result highlights the crucial role that clear and structured visual inputs play in the Octopus model, emphasizing that it significantly leverages visual cues for effective planning and task execution.

5.3 Results on Minecraft and GTA Tasks

Results on OctoMC According to the OctoMC part in Sec. 3, we designed 40 tasks, each task is operated on 2 locations so a total of 80 tasks. We set aside 10 tasks as unseen tasks and 10 tasks as seen tasks, getting 60 training tasks and 30 testing tasks in OctoMC. Similar to OctoGibson, the training data for OctoMC is collected using GPT-4. The agent, guided by GPT-4, explores the Minecraft environment and generates action plans and corresponding code based on the provided system messages, environmental cues, and objectives.

Table 4 shows that the SFT model trained on OctoMC can complete most tasks in both seen and unseen scenarios, demonstrating better performance compared to OctoGTA. However, upon analyzing the failure cases, we find that the

Table 4: Main Results for GTA and Minecraft Tasks. Despite limited training data, the Octopus still shows its good ability in task completion. In cells displaying two values, the first represents the task completion rate across the target validation task sets, while the second assesses the conceptual accuracy of the model's planning as judged by human evaluators.

Model	Vision Model	Language Model	OctoMC			OctoGTA		
			Seen Task	Unseen Task	All	Seen Task	Unseen Task	All
GPT-4	-	-	0.70 / 0.85	0.60 / 0.80	0.65 / 0.83	0.55 / 0.80	0.50 / 0.60	0.54 / 0.76
GPT-4V	-	-	0.75 / 0.85	0.70 / 0.85	0.73 / 0.85	0.58 / 0.85	0.50 / 0.70	0.56 / 0.82
EmbodiedGPT	CLIP-ViT	MPT-7B	0.30 / 0.55	0.20 / 0.60	0.25 / 0.58	0.15 / 0.38	0.20 / 0.60	0.16 / 0.42
Octopus (SFT Only)	CLIP-ViT	MPT-7B	0.30 / 0.60	0.20 / 0.70	0.25 / 0.65	0.18 / 0.48	0.20 / 0.60	0.18 / 0.50
Octopus (SFT + RLEF)	CLIP-ViT	MPT-7B	0.40 / 0.60	0.20 / 0.70	0.30 / 0.65	0.18 / 0.53	0.30 / 0.70	0.20 / 0.56

model sometimes struggles with tasks requiring precise spatial reasoning. For instance, when tasked with killing a pig, the agent may have difficulty finding the creature with the exact angle and distance. While it shows that the agent relies on visual information to navigate and interact with the environment, it also means that even with correct planning, imprecise actions can lead to failure.

Results on OctoGTA According to the OctoGTA part in Sec. 3, we designed 25 tasks. We set aside 5 tasks in the boat-related group (e.g., boat retrieval and shore return) as unseen tasks for testing only, using 2 different locations. We replicate the remaining 20 tasks for both training (8 different locations) and testing (2 locations). As a result, we have 160 training tasks and 50 testing tasks in OctoGTA. Unlike the training procedure for OctoGibson and OctoMC, the training data for OctoGTA is entirely created by the authors, as it is challenging to gather textual environmental messages in the GTA environment.

Table 4 shows that, despite having only 160 training tasks, the SFT model can complete some tasks in both seen and unseen scenarios, and RLEF also outperforms. However, upon careful examination of the failure cases, we find that the model struggles with tasks that are not straightforward. For instance, when a wall separates the player from the car, the player still finds it difficult to decide to climb the wall, even if similar cases exist in the training data. Similar to OctoMC, as illustrated in Sec. 3, when approaching certain locations, the code involves functions like `turnPlayer()` and `goForward()` rather than a simple `walkTo(location)`. Consequently, even with correct planning, imprecise actual actions can still lead to task failure.

6 Conclusion

This paper introduces Octopus, an embodied vision-language programmer designed to bridge the gap between high-level planning and real-world manipulation with programming. By open-sourcing our OctoVerse environments, dataset, and Octopus architecture, we aim to foster collaboration and innovation within the research community, paving the way for future developments in embodied vision-language programming.

Acknowledgments and Disclosure of Funding

This study is supported by the Ministry of Education, Singapore, under its MOE AcRF Tier 2 (MOET2EP20221- 0012), NTU NAP, and under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

References

1. Grand theft auto v (2014)
2. Minecraft (2023)
3. 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., Zeng, A.: Do as i can and not as i say: Grounding language in robotic affordances. In: arXiv preprint arXiv:2204.01691 (2022)
4. 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)
5. Awadalla, A., Gao, I., Gardner, J., Hessel, J., Hanafy, Y., Zhu, W., Marathe, K., Bitton, Y., Gadre, S., Sagawa, S., Jitsev, J., Kornblith, S., Koh, P.W., Ilharco, G., Wortsman, M., Schmidt, L.: Openflamingo: An open-source framework for training large autoregressive vision-language models. arXiv preprint arXiv:2308.01390 (2023)
6. Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., Clune, J.: Video pretraining (vpt): Learning to act by watching unlabeled online videos (2022)
7. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research **47**, 253–279 (Jun 2013). https://doi.org/10.1613/jair.3912, http://dx.doi.org/10.1613/jair.3912
8. Billard, A., Kragic, D.: Trends and challenges in robot manipulation. Science **364**(6446), eaat8414 (2019)
9. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
10. 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)
11. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
12. Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3d: Learning from rgb-d data in indoor environments. arXiv preprint arXiv:1709.06158 (2017)

13. Chen, J., Mu, Y., Yu, Q., Wei, T., Wu, S., Yuan, Z., Liang, Z., Yang, C., Zhang, K., Shao, W., Qiao, Y., Xu, H., Ding, M., Luo, P.: Roboscript: Code generation for free-form manipulation tasks across real and simulation (2024)
14. Chen, L., Li, B., Shen, S., Yang, J., Li, C., Keutzer, K., Darrell, T., Liu, Z.: Language models are visual reasoning coordinators. In: ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models (2023)
15. Chiang, W.L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J.E., et al.: Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. See https://vicuna. lmsys. org (accessed 14 April 2023) (2023)
16. Dai, W., Li, J., Li, D., Tiong, A.M.H., Zhao, J., Wang, W., Li, B., Fung, P., Hoi, S.: Instructblip: Towards general-purpose vision-language models with instruction tuning. arXiv preprint arXiv:2305.06500 (2023)
17. Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., Batra, D.: Embodied question answering (2017)
18. 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., Florence, P.: Palm-e: An embodied multimodal language model. In: arXiv preprint arXiv:2303.03378 (2023)
19. Evans, J.S.B.: Dual-processing accounts of reasoning, judgment, and social cognition. Annu. Rev. Psychol. **59**, 255–278 (2008)
20. Fan, L., Wang, G., Jiang, Y., Mandlekar, A., Yang, Y., Zhu, H., Tang, A., Huang, D.A., Zhu, Y., Anandkumar, A.: Minedojo: Building open-ended embodied agents with internet-scale knowledge (2022)
21. Fu, H., Xu, W., Ye, R., Xue, H., Yu, Z., Tang, T., Li, Y., Du, W., Zhang, J., Lu, C.: Rfuniverse: A multiphysics simulation platform for embodied ai (2023)
22. Fu, Z., Zhao, T.Z., Finn, C.: Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In: arXiv (2024)
23. Gao, X., Gong, R., Shu, T., Xie, X., Wang, S., Zhu, S.C.: Vrkitchen: an interactive 3d virtual environment for task-oriented learning (2019)
24. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE international conference on robotics and automation (ICRA). pp. 3389–3396. IEEE (2017)
25. Gupta, T., Kembhavi, A.: Visual programming: Compositional visual reasoning without training. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14953–14962 (2023)
26. Huang, S., Jiang, Z., Dong, H., Qiao, Y., Gao, P., Li, H.: Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. arXiv preprint arXiv:2305.11176 (2023)
27. Huang, W., Wang, C., Zhang, R., Li, Y., Wu, J., Fei-Fei, L.: Voxposer: Composable 3d value maps for robotic manipulation with language models. arXiv preprint arXiv:2307.05973 (2023)
28. Hussein, A., Gaber, M.M., Elyan, E., Jayne, C.: Imitation learning: A survey of learning methods. ACM Computing Surveys (CSUR) **50**(2), 1–35 (2017)
29. Kahneman, D.: Thinking, fast and slow. macmillan (2011)
30. Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., et al.: Ai2-thor: An interactive 3d environment for visual ai. arXiv preprint arXiv:1712.05474 (2017)
31. Li, B., Zhang, Y., Chen, L., Wang, J., Yang, J., Liu, Z.: Otter: A multi-modal model with in-context instruction tuning. arXiv preprint arXiv:2305.03726 (2023)

32. Li, C., Zhang, R., Wong, J., Gokmen, C., Srivastava, S., Martín-Martín, R., Wang, C., Levine, G., Lingelbach, M., Sun, J., et al.: Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In: Conference on Robot Learning. pp. 80–93. PMLR (2023)
33. Li, J., Li, D., Savarese, S., Hoi, S.: Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. arXiv preprint arXiv:2301.12597 (2023)
34. Lin, K., Agia, C., Migimatsu, T., Pavone, M., Bohg, J.: Text2motion: From natural language instructions to feasible plans. arXiv preprint arXiv:2303.12153 (2023)
35. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning (2023)
36. Liu, H., Li, C., Li, Y., Li, B., Zhang, Y., Shen, S., Lee, Y.J.: Llava-next: Improved reasoning, ocr, and world knowledge (January 2024), `https://llava-vl.github.io/blog/2024-01-30-llava-next/`
37. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning (2023)
38. Liu, Z., Dong, Y., Rao, Y., Zhou, J., Lu, J.: Chain-of-spot: Interactive reasoning improves large vision-language models. arXiv preprint arXiv:2403.12966 (2024)
39. Mao, H., Wang, C., Hao, X., Mao, Y., Lu, Y., Wu, C., Hao, J., Li, D., Tang, P.: Seihai: A sample-efficient hierarchical ai for the minerl competition (2021)
40. MosaicML: Mpt-7b (2023), `https://www.mosaicml.com/blog/mpt-7b`, accessed: 2023-05-23
41. Mu, Y., Zhang, Q., Hu, M., Wang, W., Ding, M., Jin, J., Wang, B., Dai, J., Qiao, Y., Luo, P.: Embodiedgpt: Vision-language pre-training via embodied chain of thought. arXiv preprint arXiv:2305.15021 (2023)
42. 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)
43. Park, J.S., O'Brien, J.C., Cai, C.J., Morris, M.R., Liang, P., Bernstein, M.S.: Generative agents: Interactive simulacra of human behavior. arXiv preprint arXiv:2304.03442 (2023)
44. Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., Torralba, A.: Virtualhome: Simulating household activities via programs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8494–8502 (2018)
45. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PMLR (2021)
46. 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)
47. Ramakrishnan, S.K., Jayaraman, D., Grauman, K.: An exploration of embodied visual exploration (2020)
48. Rana, K., Haviland, J., Garg, S., Abou-Chakra, J., Reid, I., Suenderhauf, N.: Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. arXiv preprint arXiv:2307.06135 (2023)
49. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., et al.: Habitat: A platform for embodied ai research. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 9339–9347 (2019)

50. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761 (2023)
51. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
52. Shen, Y., Song, K., Tan, X., Li, D., Lu, W., Zhuang, Y.: Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. arXiv preprint arXiv:2303.17580 (2023)
53. Surís, D., Menon, S., Vondrick, C.: Vipergpt: Visual inference via python execution for reasoning. arXiv preprint arXiv:2303.08128 (2023)
54. 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)
55. Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., Anandkumar, A.: Voyager: An open-ended embodied agent with large language models (2023)
56. Wang, H., Liang, W., Gool, L.V., Wang, W.: Towards versatile embodied navigation (2022)
57. Wu, Y., Wu, Y., Gkioxari, G., Tian, Y.: Building generalizable agents with a realistic and rich 3d environment. arXiv preprint arXiv:1801.02209 (2018)
58. Wu, Z., Wang, Z., Xu, X., Lu, J., Yan, H.: Embodied task planning with large language models. arXiv preprint arXiv:2307.01848 (2023)
59. Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., Yi, L., Chang, A.X., Guibas, L.J., Su, H.: Sapien: A simulated part-based interactive environment (2020)
60. Xie, B., Zhang, S., Zhou, Z., Li, B., Zhang, Y., Hessel, J., Yang, J., Liu, Z.: Funqa: Towards surprising video comprehension. arXiv preprint arXiv:2306.14899 (2023)
61. Yan, C., Misra, D., Bennnett, A., Walsman, A., Bisk, Y., Artzi, Y.: Chalet: Cornell house agent learning environment (2019)
62. Yu, W., Gileadi, N., Fu, C., Kirmani, S., Lee, K.H., Arenas, M.G., Chiang, H.T.L., Erez, T., Hasenclever, L., Humplik, J., et al.: Language to rewards for robotic skill synthesis. arXiv preprint arXiv:2306.08647 (2023)
63. Yuan, H., Zhang, C., Wang, H., Xie, F., Cai, P., Dong, H., Lu, Z.: Skill reinforcement learning and planning for open-world long-horizon tasks (2023)
64. Zheng, S., Liu, J., Feng, Y., Lu, Z.: Steve-eye: Equipping llm-based embodied agents with visual perception in open worlds (2023)
65. Zhou, X., Girdhar, R., Joulin, A., Krähenbühl, P., Misra, I.: Detecting twenty-thousand classes using image-level supervision. In: European Conference on Computer Vision. pp. 350–368. Springer (2022)
66. Zhu, Y., Wong, J., Mandlekar, A., Martín-Martín, R., Joshi, A., Nasiriany, S., Zhu, Y.: robosuite: A modular simulation framework and benchmark for robot learning. arXiv preprint arXiv:2009.12293 (2020)