

LangSuit-E: Planning, Controlling and Interacting with Large Language Models in Embodied Text Environments

Zixia Jia^{1,*}, Mengmeng Wang^{1,*}, Baichen Tong¹, Song-Chun Zhu^{1,2,3}, Zilong Zheng^{1,†}

¹ National Key Laboratory of General Artificial Intelligence, BIGAI

² Peking University ³ Tsinghua University

<https://github.com/bigai-nlco/langsuite.git>

Abstract

Recent advances in Large Language Models (LLMs) have shown inspiring achievements in constructing autonomous agents that rely on language descriptions as inputs. However, it remains unclear how well LLMs can function as few-shot or zero-shot embodied agents in dynamic interactive environments. To address this gap, we introduce LangSuit-E, a versatile and simulation-free testbed featuring 6 representative embodied tasks in textual embodied worlds. Compared with previous LLM-based testbeds, LangSuit-E (i) offers adaptability to diverse environments without multiple simulation engines, (ii) evaluates agents’ capacity to develop “internalized world knowledge” with embodied observations, and (iii) allows easy customization of communication and action strategies. To address the embodiment challenge, we devise a novel chain-of-thought (CoT) schema, EmMem, which summarizes embodied states w.r.t. history information. Comprehensive benchmark results illustrate challenges and insights of embodied planning. LangSuit-E represents a significant step toward building embodied generalists in the context of language models.

1 Introduction

Creating embodied agents that can perceive, think, and interact with simulated environments from the agents’ own perspective is a long-standing yet imperative challenge within the realms of robots and artificial intelligence. Recent striking progress of Large Language Models (LLMs) has ushered promising avenues for constructing autonomous agents, wherein LLMs serve as their central planners or controllers, evidenced by several intriguing demos and benchmarks within simulated environments (Reed et al., 2022; Park et al., 2023; Chevalier-Boisvert et al., 2019) or even realistic

physical worlds (Driess et al., 2023; Huang et al., 2022) across a diverse spectrum of domains, spanning from web browsing (Zhou et al., 2023a; Yao et al., preprint), language games (Park et al., 2023) and householding (Shridhar et al., 2021), etc. These language agents, reliant solely upon language descriptions as their inputs, can benefit from LLMs’ exhibited commonsense knowledge, emergent semantic reasoning and problem-solving skills. Despite the promising advances of prior LLM-as-agent tasks, it remains unclear that:

*Without perception error caused by visual models, can LLMs behave as **embodied agents** to solve complicated tasks within dynamic environments?*

Previous works mostly treat LLM agents as task planners, providing them with the global observations and all required information (e.g., entire history) in natural language to produce the next step, which contrasts the nature of chat-based language models that can potentially form their internal map and history across conversational turns. Moreover, the action spaces across tasks are specially designed and can only be evaluated within specific environments, which limits the potential capacity of evaluating the generalizability across tasks. In stark contrast, research on embodied agents and environments has gone through many successful attempts – numerous open-source embodied systems developed with realistic simulation engines, such as AI2Thor (Kolve et al., 2017b), VirtualHome (Puig et al., 2018), spurring the trend of learning embodied instruction-following challenges, such as navigation (Chattopadhyay et al., 2021), manipulations (Ehsani et al., 2021), question answering (Gordon et al., 2018), etc. Models learned within these environments, nevertheless, struggle with their dependency on simulation engines and are therefore limited to specific tasks. Other prominent research works such as PaLM-E (Driess et al., 2023) and RT (Zitkovich et al., 2023) sought to powerful vision-language models (VLMs) for embod-

*Equal contribution

†Correspondence to Zilong Zheng <zlzheng@bigai.ai>

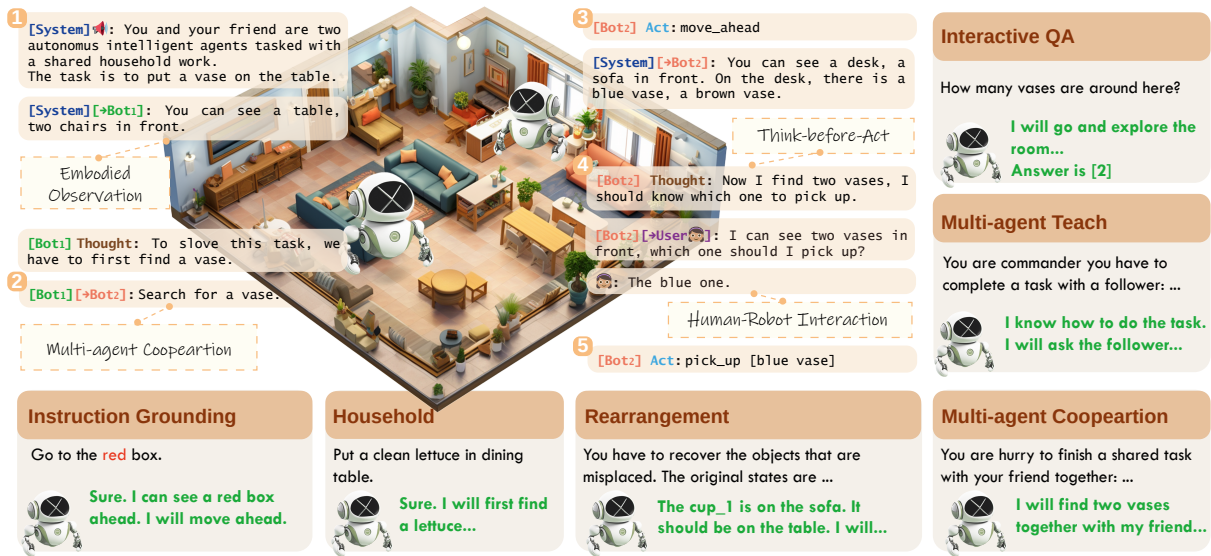


Figure 1: **Task illustration of LangSuit-E.** Top-Left: a typical example of a cooperative task between two agents. The agents are entirely blind (*i.e.*, without visual perception) and can only obtain embodied information from the system. Around: Exemplar tasks supported by LangSuit-E.

ied scene perception and language models for low-level control. However, these works can only be applied to constrained environments and require large volumes of data for pre-training.

To fill the gap and step towards general embodied intelligence, we introduce **LangSuit-E**, a systematic and simulation-free testbed for evaluating LLMs’ capabilities across different embodied tasks in textual embodied worlds. As listed in Tab. 1, LangSuit-E showcases 6 representative embodied tasks, with variants of the number of rooms, the number of agents, the action spaces of agents (whether they can communicate with each other or ask humans). Compared with prior LLM-based agent testbed, LangSuit-E presents superior benefits on building and benchmarking LLMs’ embodied capabilities. **First**, due to the nature of textual worlds, LangSuit-E incorporates a general yet fully customizable indoor environment that can easily adapted to most simulated embodied environments, (*e.g.*, AI2Thor (Kolve et al., 2017b), VirtualHome (Puig et al., 2018)), without resorting to variants of simulation engines and benchmark versions. **Second**, LangSuit-E only presents embodied observations to agents at each step. A typical example is shown in Figure 1, where only in-front observations within certain distances (unblocked by walls) are presented to agents at each step. Prior study (Huang et al., 2022) has shown that language models can potentially form “internalized world knowledge” through thinking in language. For in-

stance, they shall potentially know that executing “turn_right by 90 degrees” four times shall remain in the same direction as the beginning. Such capacity could potentially enable LLM agents to reason over action histories and observations to plan for the next. **Third**, the agents in LangSuit-E are fully-customizable w.r.t. their action spaces and communicative capabilities, *i.e.*, one can easily adapt the communication and acting strategy from one task to another.

To evaluate the current prevalent LLMs’s performance on LangSuit-E, we derive a comprehensive benchmark across 6 different tasks. Of note, annotating or generating golden trajectories is commonly labor-intensive and time-consuming. In LangSuit-E, we can easily form a benchmark by leveraging and modifying existing annotations, a by-product benefit of pursuing such a general textual embodied world. An underlying hypothesized benefit is that knowledge learned from one task can also enhance the performance of the others.

Lastly, inspired by grid cells in neural science (Banino et al., 2018) and the “lost-in-middle” observation of long-context language models (Liu et al., 2023a), we propose EmMem, an **Embodied** planning strategy with an emergent embodied **Memory**. Specifically, the language model learns to summarize and predict their embodied states in an in-context manner given long-context trajectories and observations. Results on the Household task present the effectiveness of the EmMem strategy.

Table 1: **Statistics of Tasks Integrated in LangSuit-E.** We provide a unified and simulation-free environment that can easily adapt to most embodied tasks and benchmarks.

Task Type	Simulator/Env.	# of Scenes	# of Tasks	# of Actions	Multi-Room	Multi-Agent	Communicative
Instruction Grounding (IG)	Mini Grid/BabyAI (2019)	105	500	6	✓	✗	✗
Rearrangement	AI2THOR/Rearrangement (2021)	120	500	8	✗	✗	✗
Interactive QA (IQA)	AI2THOR/IQA (2018)	30	3000	5	✗	✗	✓
Household	AI2THOR/ALFred (2020a)	120	506	12	✗	✗	✗
Multi-agent Household (MA-Household)	AI2THOR/TEACh (2022)	120	200	13	✗	✓	✓
Multi-agent Watch-and-Help (MA-WAH)	VirtualHome/Co-WaH (2023)	2	50	6	✓	✓	✓

In summary, this paper makes the following contributions: i) we deliver a systematic and useful testbed with 6 exemplar tasks for testing and training LLMs as embodied agents; ii) We introduce EmMem, a novel prompting strategy for embodied tasks; iii) We show substantive benchmark results and call for future exploration in LangSuit-E.

2 Related Work

Evaluating LLM as Autonomous Agents The concept of LLM-as-Agent has triggered the development of many systematic and interactive environments that allow for textual interactions and show well-aligned language groundings in visual or physical modalities. TextWorld (Côté et al., 2018) is one of the earliest testbeds that presented a text-based game sandbox for training and testing reinforcement learning (RL) agents. Based on this framework, ALFWorld (Shridhar et al., 2021) proposed an embodied challenge that aligns the ALFRED (Shridhar et al., 2020a) benchmark to TextWorld. However, this environment still requires a simulator to generate text descriptions and simplifies the embodied challenge by presenting full world observations to agents (Fig. 2). More recently, enhanced by harnessing the capacities of chat-based LLMs, a variety of language agents targeting a spectrum of different tasks have been demonstrated. For example, WebShop (Yao et al., preprint), Mind2Web (Deng et al., 2023) and WebArena (Zhou et al., 2023a) present environments on open-domain web shopping and browsing; Agents (Zhou et al., 2023b) design a library that supports long-short term memory; Agent-Bench (Liu et al., 2023b) presents a comprehensive evaluation of 25 different LLMs across 8 environments. In addition to single-agent settings, recent advances also show emergent behaviors in multi-agent LLM systems (Gong et al., 2023; Park et al., 2023; Chen et al., 2023; Wang et al., 2023b). However, most of these tasks over-simplify the agent’s observations and actions by presenting with full knowledge of the world’s status.

LLMs for Embodied Tasks and Robotics The rise of LLMs has spurred many inspirations on conventional embodied and robotics systems. (Vemprala et al., 2023) presented an empirical success of using ChatGPT in zero-shot robotics controls. SayCan (Ahn et al., 2022) and LLM-Planner (Song et al., 2022) employ visual navigation to collect information in the house for the challenging grounded plan generation. Another thread of works presents attempts on multimodal embodied environments by leveraging a visual-language pretraining model for world perception and LLMs for planning. For instance, many works leverage LLMs to enhance agent ability in the Minecraft environment (Wang et al., 2023a; Zhu et al., 2023; Reed et al., 2022; Baker et al., 2022; Fan et al., 2022). Wu et al. (2023b) construct a multimodal dataset containing triplets of indoor scenes, instructions, and action plans, where they employ the AI2-THOR simulator (Kolve et al., 2017a) to expand indoor scenes as environments and leverage GPT-3.5 to automatically annotating action plans. However, due to the imperfect multimodal understanding and dynamic scene environments, these tasks are limited to single-agent settings and simple manipulations and navigation in domain-specific environments. Differently, LangSuit-E presents a simulation-free and embodied framework that allows a variety of environments and agent settings.

Planning and Acting with LLMs Inspired by the success of chain-of-thought (CoT) on language reasoning tasks, many works propose to enhance the planning and acting capabilities of LLMs in a similar manner. ReAct (Yao et al., 2022) proposes to combine CoT reasoning and actions to enhance the efficiency of language models. Dasgupta et al. (2023) investigate a Planner-Actor-Reporter agent paradigm, where the Planner is equipped with a language model to perform reasoning. Wu et al. (2023a) propose a Plan, Eliminate and Track framework, aiming to mitigate constraints, such as fine-tuning inefficiency and bias from pre-training, of LLM with transformer architecture. Related to us,

Table 2: Examples of different action settings in LangSuit-E. Refer to Appendix for the complete set.

Action Type	Description
<i>Low-level actions</i>	
moveAhead	Move ahead by 1 step.
turnLeft	Turn left by 90 degrees.
turnRight	Turn right by 90 degrees.
pickUp [obj]	Pick up an object identified by [obj].
put [obj, recep]	Put the object [obj] into the receptacle [recep].
open [obj]	Open object [obj] if it is openable.
close [obj]	close object [obj] if it is closeable.
stop [answer]	Stop making actions and conclude the final answer in [answer].
...	...
<i>High-level actions</i>	
goto [loc]	Go to a location that is in current observation.
...	...
<i>Communicative actions</i>	
chat [message]	Send messages to other agents.
ask [message]	Send queries to users.

Pan et al. (2023) transfers visual perception from the virtual environment to language descriptions, then prompts LLM to solve the specific navigation tasks only leveraging language descriptions. But different from them, we provide a unified embodied environment that is capable of real-time feedback and integrates various embodied tasks.

3 The LangSuit-E Testbed

The goal of LangSuit-E is to provide a unified testbed thereby evaluating the diversified ability of LLM-as-Agents in a dynamic embodied environment with textualized perceptual representation and realistic physical limitation. Moreover, by unifying various embodied tasks in one environment, agents could improve the ability of knowledge transfer and construct consistent world cognition from interactions with the environment to solve different kinds of tasks.

3.1 Generic Embodied Task Planning

LangSuit-E aims to construct a general textual embodied world by defining a lightweight and simulation-free environment system. The embodied task of LangSuit-E can be formulated as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{U}, \mathcal{O})$, which encompasses embodied agent state space \mathcal{S} , action space \mathcal{A} , deterministic transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, reward function \mathcal{R} , task instruction space \mathcal{U} and embodied observation space \mathcal{O} .

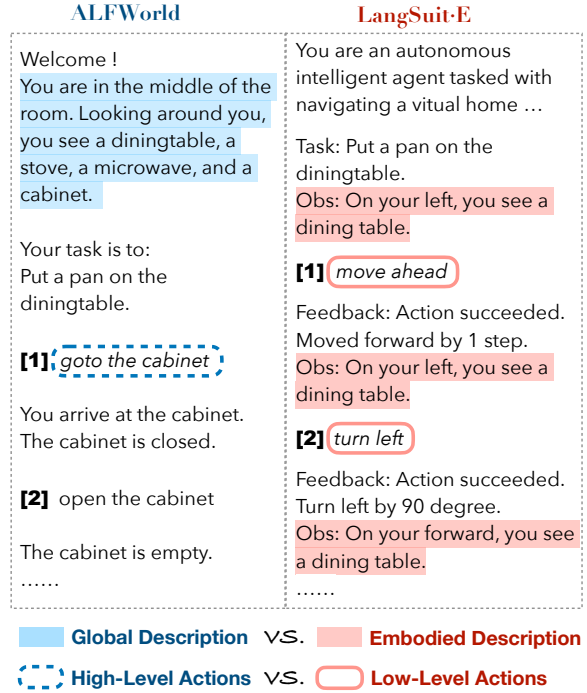


Figure 2: Comparison of our Embodied Environment Description with ALFWORLD (Shridhar et al., 2021).

Embodied States The state refers to the representation of the environment that an agent is in at a given time. LangSuit-E encompasses diverse indoor scene layouts. The state of our embodied setting includes the indoor arrangement, object information, the history of the previous action decisions and executive outcomes, the overall target or programmatic sub-goals, the limitations that must be followed, and so on.

Embodied Observations To simulate real-world conditions, the environment is partially observable, which means that an agent is not capable of perceiving the entire world or environment states. Based on this, LangSuit-E provides the embodied observations of partial objects in the environment with full customization of each agent’s view range. In particular, LangSuit-E includes a module dedicated to describing environmental objects that are perceptible within an agent’s uniquely configured visual scope using natural language. For instance, the language expressions could be “You can see a red ball on your right” or “You can see nothing ahead”.

Embodied Actions LangSuit-E incorporates a wide spectrum of executable embodied actions, covering most actions typically encountered within various embodied environments. The agent decides to execute a specific action in each time step according to the embodied state and its internal world

knowledge. Tab. 2 presents an enumeration of the action types that are currently accommodated in the LangSuit-E. Refer to Tab. 9 in Appendix for the complete action set. Furthermore, LangSuit-E exhibits substantial flexibility by allowing for the customized configuration of action scopes. For example, the movement distance associated with the `move_ahead` action can be readily modified by adjusting the relevant hyperparameters, thereby facilitating tailored adjustments to suit specific embodied task requirements.

Feedback LangSuit-E environment conveys feedback through the use of natural language descriptions. This feedback is automatically generated by our system by evaluating the feasibility of the action predicted by LLMs in the context of the current state. The evaluation takes into account physical constraints imposed by both the agent and the environment, such as the agent’s grasping range or environmental obstacles, ensuring that the action aligns with the agent’s physical capabilities and the environmental context; refer to Fig. 3 and Appendix for examples.

3.2 Communications in LangSuit-E

The platform additionally provides multi-agent and human-agent communication channels to enable multi-agent cooperation and human-in-the-loop tasks. The *Communicative actions* in Tab. 2 are additional actions to be appended for this capacity.

(Multi-)Agent Interactions Multi-agent cooperations are widely explored in many recent LLM-as-agent tasks (Zhou et al., 2023b; Park et al., 2023). In LangSuit-E, the agents are fully customizable regarding their action spaces and communicative capabilities, *i.e.*, one can easily adapt the communication and acting strategy from one task to another. See Fig. 1 for a typical example.

Human Communication LangSuit-E supports human-agent interactions through two primary mechanisms: firstly, it empowers humans to control an agent or play the role of the agent to cooperate with other agents. Secondly, it affords agents the capability to seek clarification from humans in instances of environmental or instructional ambiguity, enabling humans to offer assistance by imparting pertinent information via a communication protocol, thus enhancing task completion.

3.3 Benchmark LangSuit-E Tasks

We integrate a diverse range of embodied tasks within the unified LangSuit-E testbed. Tab. 1 demonstrates the specific categories of tasks that have been accommodated within LangSuit-E. We carefully select the six tasks to illustrate how our testbed can be applied to single-agent embodied understanding and planning (Instruction Grounding, Interactive QA, Household and Rearrangement), multi-agent cooperation (MA-WAH), multi-agent teaching (MA-Household), each of which represents a significant challenge within the current embodied community. We re-implement all tasks in our embodied environments to reach the objective of a unified simulation-free testbed. Notably, LangSuit-E has an inherent capacity for facile expansion and adaptation to accommodate additional embodied tasks with minimal resource consumption, for the reason of the wealth meta-capabilities it possesses, encompassing easy customization of communication and action strategies and the ability to support interactions involving multiple agents, among others.

Task Instructions Task instruction provided to an agent focuses on the characteristics and constraints of tasks with a particular type, involving customized descriptions of the specific type, action spaces of tasks, and optional few-shot examples to show the diverse execution processes of some tasks under the same type.

LangSuit-E provides a template for the automated generation of diverse task instructions. By filling out customized task descriptions, definitions of action spaces pertinent to a particular task type, or the incorporation of few-shot illustrative examples, LangSuit-E is capable of generating distinct sets of instructions that serve as prompts for LLMs regarding different tasks. The tasks that the current version of LangSuit-E supports:

Task I. Instruction Grounding (IG) (Chevalier-Boisvert et al., 2019) provides synthetic natural-looking instructions (*e.g.*, “put the red ball next to the box on your left”), requiring the agent to navigate the world (including unlocking doors) and move objects to specified locations.

Task II. Rearrangement (Weihs et al., 2021) involves moving and modifying (*i.e.*, opening/closing) randomly placed objects within a room to obtain a goal configuration.

Task III. Interactive QA (IQQA) (Gordon et al., 2018) is the task of answering questions

that require the agent to interact with a dynamic environment, which contains three different question types: Existence questions (“Is there an apple in the kitchen?”), Counting questions (“How many forks are present in the scene?”), and Spatial Relationship, called Contains questions (“Is there lettuce in the fridge?”).

Task IV. Household (Shridhar et al., 2020a) tasks contain both high-level goals like “Rinse off a mug and place it in the coffee maker.” and low-level language instructions like “Walk to the coffee maker on the right.”

Task V. Multi-agent Teach (MA-Teach) (Padmakumar et al., 2022) tasks should be completed with the cooperation of two agents. A Commander with access to oracle information about a task communicates in natural language with a Follower. The Follower navigates through and interacts with the environment to complete tasks varying in complexity from “make coffee” to “prepare breakfast”, asking questions, and getting additional information from the Commander.

Task VI. Multi-agent Cooperation tasks focus on evaluating the cooperation and communication ability of multiple agents to accomplish shared goals. We follow C-WAH (Zhang et al., 2023) to use a watch-and-help (WAH) configuration in this task. Specifically, all agents shall negotiate with each other to break down the task based on their individual observations and capabilities. Different from C-WAH, all agents in this task can only perceive their own embodied observations.

Gold Trajectories Generation We have implemented an extra expert module for task I-IV (the multi-agent settings require agent interactions), dedicated to generating gold trajectories and affording us the capability to construct training datasets that may be subsequently employed for the potential fine-tuning of LLMs. Given a task without gold trajectories, our expert module automatically produces a reasonable and executable trajectory, including every-step actions. Specifically, we first analyze a list of objects necessitating manipulation according to the answer or the final state of the task. Second, we construct a list of sub-tasks corresponding to each operation of an object and carefully devise the execution sequence of sub-tasks. Finally, we adopt the A-star algorithm to chart the shortest paths between the dynamic agent location (the

agent’s location after finishing the last sub-task) and the target object for each sub-task. As a consequence, our expert module outputs an exhaustive sequence of actions meticulously crafted to fulfill the task at hand.

3.4 Evaluation Metrics

We follow metric design schemas as in (Weihs et al., 2021; Shridhar et al., 2020a) to provide the following evaluation metrics:

- **Success Rate (SR)**: the ratio of tasks in which an agent achieves the target goal or state.
- **Goal-Conditioned Success Rate (Goal-SR)**: the sub-goal completion ratio of a task, averaged across all the tasks of the same type.
- **Misplaced Rate (Misplaced)**: the misplaced objects in the *Rearrangement* task, which computes the ratio of misplaced objects at the end of the task, denoting better performance with a lower value.
- **Fixed Strict Rate (Fixed Strict)**: the correctly reset objects in the *Rearrangement* task, which computes the ratio of correctly reset objects at the end.
- **Accuracy (Acc)**: the proportion of correct answers compared to the total number of questions. The metric solely evaluates on the *IQA* task.
- **Average Step (Avg Steps)**: the average number of actions an agent executes to complete all the tasks.

4 EmMem: Interactive Planning with Embodied Memory

One *de facto* paradigm of prompting LLMs for planning tasks is to leverage chain-of-thought (CoT) (Wei et al., 2022) strategy which carefully designs and selects cases of reasoning traces as in-context examples. ReAct (Reason+Act) (Yao et al., 2022), one prior success in a couple of reasoning and planning tasks, generates both *reasoning traces* and actions through an extra **Thought** process. Following this, Reflection (Shinn et al., 2023) proposes to verbally reflect on task feedback signals, then maintain reflective text in a memory buffer in subsequent trials. However, all these strategies failed to reveal satisfactory results in our embodied environments; refer to Fig. 3 for typical failures.

We present EmMem, a novel embodied planning paradigm that reasons and summarizes the agent’s current embodied memory in an in-context and

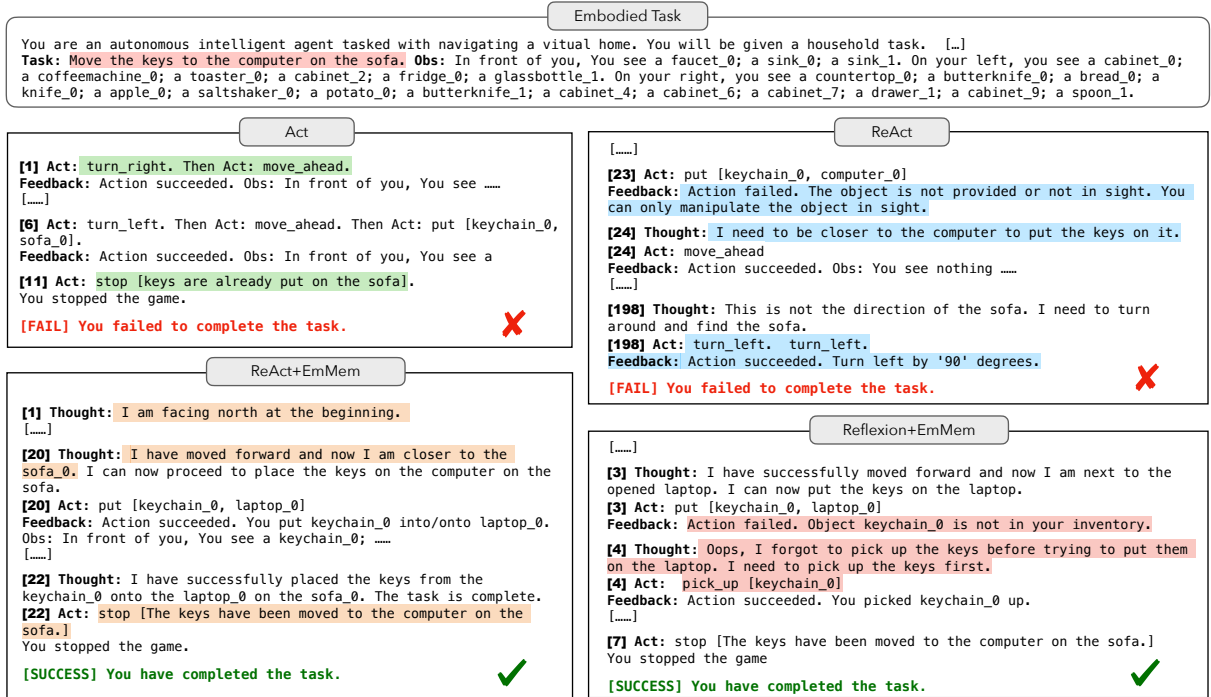


Figure 3: A case study for four prompt strategies on Household. Act: The model predicts the next action only. ReAct+EmMem: Our EmMem strategy with ReAct strategy (Reason+Act) (Yao et al., 2022). Reflexion+EmMem: Our EmMem strategy with Reflexion strategy (Shinn et al., 2023) (1 additional trail of verbal reflection). Because Reflexion will try an additional trail if the task fails and the summarization of the whole last trail will be input to the new trail planning through the Task prompt, we do not show the case in this figure. More details and the prompt instructions for these strategies can be found in Appendix G.

emergent manner. Specifically, the LLM agent has to explicitly predict and summarize their current embodied state before proceeding to the actual planning and action prediction step. We provide two underlying rationales for such prompts: **First**, as for mammals’ embodied navigation, neurons like grid cells integrate path information and are thought to be critical in self-motion (Banino et al., 2018; Hafting et al., 2005). The EmMem, auto-regressively in principle, produces integrated embodied information as grid cells. **Second**, recent literature has observed the lost-in-middle performance for long-context language models (Liu et al., 2023a). Our EmMem enhances the “middle” information in a residual-like approach (He et al., 2016). Fig. 3 illustrates a case study of EmMem and other strategies.

5 Experiments

5.1 Setup

We evaluate the ability of popular LLMs as agents to solve embodied tasks in our embodied textual environment. It is crucial to note that our testbed does not provide any global information, necessitating that agents mobilize more capabilities in

the more realistic and complicated embodied environment to execute tasks, including comprehension of embodied description, retention of historical exploration within the environment, and self-awareness of its current state. We mainly use¹ gpt-3.5-turbo (with temperature being 1) to evaluate on LangSuit-E. **Notably**, for API-cost economy, We evaluated 100 randomly sampled tasks for IG, Household, and Rearrangement, along with 50 randomly sampled tasks for each sub-task of IQA (Exists, Contains, and Counts), MA-Household, and MA-WAH. We conduct experiments both in the low-level (fine-grained actions) and high-level action settings. The evaluation metrics adopted mainly are introduced in Section 3.4.

5.2 Main Results and Observations

In this section, we summarize the empirical results and focus on addressing the following questions:

Q1: How do previous CoT strategies perform in LangSuit-E? Fig. 3 and Tab. 3 show the qualita-

¹We did not fully test some open-source LLMs with small sizes, such as LLaMA, because we made a preliminary attempt to utilize LLaMA 2-7B to solve Instruction Grounding and IQA tasks but found it failed in all tasks

Table 3: Evaluation results on the *test* set of single-agent task types in LangSuit-E. The evaluation results on Household task in terms of Path Weighted Metrics are in Appendix D.

Model	IG		Rearrangement				IQA				Household		
	SR.	Avg Steps.↓	SR.	Misplaced.↓	Fixed Strict.	Avg Steps.↓	Exists Acc.	Contains Acc.	Counts Acc.	Avg Steps.↓	SR.	Goal-SR.	Avg Steps.↓
<i>Low-Level Actions Setting</i>													
Act	13.0	28.2	8.0	115.3	17.6	12.4	14.9	36.0	10.9	16.7	9.0	21.5	22.8
ReAct	12.0	25.3	5.0	117.9	12.3	8.4	8.9	18.3	2.1	13.2	14.0	26.7	44.7
ReAct-EmMem	11.0	22.8	6.0	108.3	13.7	9.5	48.0	58.0	6.0	14.8	10.0	26.3	28.5
Reflexion	20.0	29.9	8.0	99.3	14.3	13.7	62.0	72.0	38.0	23.7	19.0	34.6	28.6
Reflexion-EmMem	16.0	36.8	10.0	100.4	15.7	12.5	60.0	86.0	72.0	20.7	14.0	32.0	35.1
<i>High-Level Actions Setting</i>													
Act	77.0	6.9	5.0	112.3	17.2	7.0	24.0	76.0	30.0	10.3	23.0	37.8	16.2
ReAct	84.0	14.3	7.0	111.2	15.2	6.9	44.0	88.0	18.0	15.1	22.0	42.0	21.4
ReAct-EmMem	85.0	16.9	6.0	109.7	11.9	9.0	32.0	80.0	14.0	13.1	32.0	47.8	35.2
Reflexion	86.0	14.4	8.0	102.7	17.5	6.5	44.0	90.0	22.0	9.9	33.0	53.0	27.7
Reflexion-EmMem	86.0	15.7	8.0	100.22	16.2	8.9	42.0	80.0	20.0	16.6	39.7	53.3	34.1

Table 4: Evaluation results on the *test* set of multi-agent task types in LangSuit-E.

Model	MA-Household		MA-WAH	
	SR.	Goal-SR.	SR.	Goal-SR.
<i>High-Level Actions Setting</i>				
Act	18.00	44.80	36.00	70.17
ReAct	18.00	60.63	38.00	78.20
ReAct-EmMem	18.00	64.74	46.00	76.13
Reflexion	18.00	66.58	56.00	88.43
Reflexion-EmMem	20.00	65.81	64.00	90.37

tive and numerical results of previous CoT methods on single-agent tasks. Our observations can be summarized as follows: ❶ *Act-only vs. ReAct vs. Reflexion*: The *Act* can not strictly follow our desired format (it generates two actions in each step, where we only require the first one), and it predicts the *stop* action even though the task is not completed. Differently, *ReAct* has the capability of considering the correct plans after the action failed, but explores more steps, resulting in exceeding our step limit (in the low-level action setting on Household). Notably, *ReAct* drops the performance seriously in the IQA tasks in low-level action setting. A possible reason is that CoT causes more severe hallucination problems, as illustrated in [Vu et al. \(2023\)](#). The *Reflexion*, which goes through more trials and self-reflection process, showcases significantly better performance on most task types. And equipped with *EmMem*, the performances are further improved in some cases. ❷ *Low-level action vs. high-level actions*: We compare the high-level action sets with low-level actions introduced in Tab. 2. In a high-level setting, an agent does not need to know how to perform low-level action sequences to achieve a goal place/object. Empirically and intuitively, we can observe that high-level actions make embodied tasks

relatively easy to solve (fewer steps and higher SR.), which indicates the challenges of our fully embodied action setting. Although prompts with **Thought** (*Reflexion*, *EmMem*) increase the average execution steps of GPT-3.5, the performances of GPT-3.5 are improved in most cases. It is surprising that the **Thought** process drops the Fixed Strict scores in the low-level actions setting on Rearrangement.

Q2: How does *EmMem* help in embodied planning tasks?

As shown in Tab. 3 and case results Fig. 3, with reasoning and summarizing the current embodied state in the *ReAct+EmMem* setting, an agent is more clear about the state of itself and the execution process of a task, hereby successfully carrying out the task; With both reflexion and memory (*Reflexion+EmMem*) strategies, an agent can recognize its mistakes quickly and reformulate following plans efficiently. We also observe that *EmMem* is more useful under the low-level action settings compared with high-level ones. Such observation is in line with the intuition that low-level settings can benefit more from embodied memory.

Q3: How can human-in-the-loop be helpful?

Previous works have shown that appropriate human-agent communication can substantially improve the efficiency in embodied grounding ([Huang et al., 2022](#)). In LangSuit-E, we also incorporate the ask-for-help setting to obtain human preferences and selection, especially when it comes to ambiguous task instructions (a qualitative demonstration in Fig. 4).

Q4: Does the temperature highly influence the *EmMem* performance?

We conducted experiments with different temperature settings for GPT-3.5 and reported the corresponding standard deviations in Tab. 5. We have observed consistent

Table 5: Evaluation results of varying temperatures (**Temp**) on Household using Reflexion-EmMem strategy in high-level actions setting.

Temp	0.5	1.0	1.5
SR.	40.00±3.46	39.67±0.58	40.67±0.58
Goal-SR.	52.47±3.71	53.30±2.43	52.86±1.13

Task: Put the gold vase into the white safe.
 Obs: You see nothing. You can try to take action like ask, move_ahead, turn_left or turn_right

[1] Thought: To complete the task, I need to find the gold vase and the white safe. Then I can put the gold vase into the white safe. I don't have any information about the location of the gold vase or the white safe. I should ask for help to find them.

[1] Act: ask [Where is the gold vase?]

Feedback: Human response: On your left.

[2] Thought: Since the gold vase and the white safe are not in my current view, I need to explore the environment to find them.

[2] Act: turn_left

Feedback: Action succeeded. Turn left by '90' degrees.
 Obs: On your left, you see a drawer_1

Figure 4: Demonstration of human communications.

performance across various temperature settings. Our hypothesis is that the CoT strategy, coupled with exemplar cases, plays a crucial role in guiding LLMs to follow a consistent reasoning trajectory. This observation underscores the effectiveness of our approach in maintaining stability and reliability in the model’s performance.

Q5: How does the EmMem perform in the Multi-agent settings? We only use high-level actions in multi-agent tasks to reduce task complexities. We observe that all API-based agents maintain the capacity of multi-agent cooperations with relatively high Goal-SRs. We did not report low-level results as there is barely a successful case. As demonstrated in Tab. 4, EmMem achieves consistent improvement, except for the **Goal-SR** metric in MA-Household with Reflexion. Note that GPT-3.5 achieves high performance in MA-WAH tasks in most of the prompt strategies, because the feedback of each action provided in MA-WAN contains the self-state of agents. It is consistent with our intuition for designing EmMem: explicitly predicting and summarizing agents’ current embodied state before proceeding to the actual planning and action prediction step is beneficial. It is exciting that even with the self-state of agents the environment provided, EmMem inspiring the self-awareness of agents’ current stat still further improves the performance.

6 Conclusion

We introduce LangSuit-E, a lightweight and simulation-free testbed in textual embodied worlds. Designed to be fully embodied, LangSuit-E aims to test LLM agent’s embodied planning capacity without errors from visual perceptions. To further address the embodiment challenge, we devise a novel CoT schema, EmMem, which summarizes embodied states w.r.t. history information. Although benchmark results indicate the effectiveness of EmMem, there still remains a huge performance gap compared with SFT results on individual tasks. As such, we call for future exploration of embodied generalists on LangSuit-E.

Limitation

In this work, we endeavor to investigate how LLM performs and understands embodiment in a generic embodied textual setting. Nevertheless, our platform maintains the potential to be bridged with most simulators via additional visual language models (VLMs). Due to the page limitation and workload capacity, it remains unclear whether such linkage would be a successful attempt, even if we have obtained a powerful LLM embodied agent. We would like to leave it as a strong inclination for future research on generic embodied multimodal agents.

Moreover, the high demand of LLM API calls leads to a substantial querying price for experiments. Preliminary studies show that open-source language models, especially small models (e.g., LLAMA 2-7B), are unable to make appropriate embodied predictions in our environment, no matter what kind of prompting strategy is used. Nevertheless, it would be more than valuable to investigate how open-source language models can be trained or finetuned in the LangSuit-E environment.

Additionally, following previous works (Yao et al., 2022), we mainly use GPT-4 and gpt-3.5-turbo for all experiments. Though we have observed performance boosts with a more powerful model, whether all observations and analyses are model-dependent remains unknown. Intuitively, our framework is model-agnostic and can be generalized to all embodied tasks. We are more than willing to experiment on more APIs once we have the access.

Lastly, we didn’t go into too much on human communications and multi-agent tasks. Nevertheless, our platform maintains the capabilities to be

generalized to multi-agent settings and human communicative settings (all you need is to update the configuration of agents). As such, learning and experimenting with multiple embodied agents in LangSuitE could also be a valuable future exploration trend

Ethics Statement

In this work, we aim to deliver a generic, lightweight and simulation-free testbed for general embodied planning. By leveraging large language models, the main purpose of this research is to benefit the embodied AI and language reasoning community. Moreover, we devise a novel reasoning paradigm, EmMem, which grants agents with emergent embodied memories. Going beyond embodied understanding, we believe the CoT paradigm is general and can be generalized to more reasoning and planning tasks.

Acknowledgements

The authors thank Dr. Wentao Ding for his great contributions in refining code repository and Mrs. Chen Zhen for her design of illustrative teasers. This work presented herein is supported by the National Natural Science Foundation of China (62376031).

References

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

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:24639–24654.

Andrea Banino, Caswell Barry, Benigno Uribe, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. 2018. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433.

Prithvijit Chattopadhyay, Judy Hoffman, Roozbeh Mottaghi, and Aniruddha Kembhavi. 2021. Robustnav: Towards benchmarking robustness in embodied navigation. In *Proceedings of the IEEE/CVF Interna-*

tional Conference on Computer Vision, pages 15691–15700.

- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023. Agent-verse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.
- Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations (ICLR)*.
- Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.
- Ishita Dasgupta, Christine Kaeser-Chen, Kenneth Marino, Arun Ahuja, Sheila Babayan, Felix Hill, and Rob Fergus. 2023. Collaborating with language models for embodied reasoning. *arXiv preprint arXiv:2302.00763*.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*.
- Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.
- Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. 2021. Manipulathor: A framework for visual object manipulation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4497–4506.
- Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:18343–18362.
- Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. 2023. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*.
- Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. 2018. Iqa: Visual question answering in interactive environments. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

- Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser. 2005. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.
- Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. 2017a. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*.
- Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017b. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023a. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023b. Agent-bench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*.
- Aishwarya Padmakumar, Jesse Thomason, Ayush Srivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. 2022. TEACH: Task-driven Embodied Agents that Chat. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, pages 2017–2025.
- Bowen Pan, Rameswar Panda, SouYoung Jin, Rogerio Feris, Aude Oliva, Phillip Isola, and Yoon Kim. 2023. Langnav: Language as a perceptual representation for navigation. *arXiv preprint arXiv:2310.07889*.
- Joon Sung Park, Joseph C O’Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.
- Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8494–8502.
- Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020a. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020b. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10740–10749.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2022. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088*.
- Sai Vemprala, Rogerio Bonatti, Arthur Buckner, and Ashish Kapoor. 2023. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res.*, 2:20.
- Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, et al. 2023. Freshllms: Refreshing large language models with search engine augmentation. *arXiv preprint arXiv:2310.03214*.
- Guangzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.
- Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei Wang, Shiji Song, and Gao Huang. 2023b. Avalon’s game of thoughts: Battle against deception through recursive contemplation. *arXiv preprint arXiv:2310.01320*.

- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.
- Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. 2021. Visual room rearrangement. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Yue Wu, So Yeon Min, Yonatan Bisk, Ruslan Salakhutdinov, Amos Azaria, Yuanzhi Li, Tom Mitchell, and Shrimai Prabhumoye. 2023a. Plan, eliminate, and track—language models are good teachers for embodied agents. *arXiv preprint arXiv:2305.02412*.
- Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. 2023b. Embodied task planning with large language models. *arXiv preprint arXiv:2307.01848*.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. preprint. Webshop: Towards scalable real-world web interaction with grounded language agents. In *ArXiv*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. 2023. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*.
- Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023a. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.
- Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, et al. 2023b. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.
- Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.
- Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *7th Annual Conference on Robot Learning*.

Table 6: Time complexity comparison of simulated environments with LangSuit-E. **Omni** is the abbreviation of OmniGibson (OmniVerse) which is a physical-based simulator, and **AI2Thor** (Unity) takes only symbolic actions.

Env. (Engine)	Omni	AI2Thor	LangSuit-E
Time Int (ms)	67.90	7.81	1.89

A The motivation of the design for LangSuit-E

First, we want to design our unified textual environment being general, embodied, interactive, and communicative, thereby containing most of the typical embodied task categories, such as navigation, rearrangement, human interaction, multi-agent cooperation, and so on. Therefore, we transfer typical embodied tasks, carefully selecting action sets that are both suitable for our environment and the original task. The differences between our benchmark and original embodied tasks are that our environment is fully embodied and the actions are low-level (fine-grained atom actions).

Second, As the previous platform generally gives the global description of the environment, trained models in the simulated environment are difficult to apply to real environments. Our platform simulation more realistic situation that agent has a limited view scope and action scope.

Third, The simulation-free platform makes various embodied tasks conveniently and easily performed in a unified fashion.

B Data and Code Release

All code and data will be publicly available for use. The data and code are under MIT license. All resources can only be used for research purposes. More details of the construction process of LangSuit-E can be found in the codebase.

C Time complexity comparison

We compare the computational cost for one-step execution/rendering on simulated environments vs. LangSuit-E. The time interval (**Time Int**) counts from receiving one LLM inference results to the next query to LLM (thus the latency caused by API-call is not counted). We run all experiments on an Intel Core i9 with 64GB RAM. Results are shown in Tab. 6. Our LangSuit-E demonstrates significant systemwide efficiency improvements.

Table 7: Evaluation results on Household task with Path Weighted Metrics.

Model	Household	
	SR.	Goal-SR.
<i>Low-Level Actions Setting</i>		
Act	6.65	21.94
ReAct	9.50	24.71
ReAct-EmMem	7.19	21.13
Reflexion	14.89	25.73
Reflexion-EmMem	11.19	22.77
<i>High-Level Actions Setting</i>		
Act	22.38	37.13
ReAct	18.64	35.52
ReAct-EmMem	28.27	43.02
Reflexion	28.23	40.00
Reflexion-EmMem	34.45	43.80

D Path Weighted Metrics

We additionally utilized Path Weighted Metrics (Shridhar et al., 2020b) to calculate the path weighted score on Household task. This weighted score penalizes results that complete the task but take a longer number of steps. The introduction of Path Weighted Metrics allows the evaluation to focus not only on task completion but also on efficiency and path length, thereby providing a more comprehensive assessment of the model’s performance. The path weighted score p_s for metric s is given as:

$$p_s = s \times \frac{L^*}{\max(L^*, \hat{L})} \quad (1)$$

where \hat{L} is the number of actions the agent took in the episode, and L^* is the number of actions in the expert demonstration.

As can be seen from the Tab. 7, after using the path weighted score, the performance on the **Act** strategy does not decline significantly. However, for other tasks, the agent is encouraged to think more, leading to an increase in the number of steps and consequently greater penalties, resulting in a more noticeable decline.

E Customized configuration of an agent.

LangSuit-E supports the customized configuration of agents. It allows the configuration of prompt strategies, the initialization position of agents, inventory capacity, focal length, manipulate distance and view distance. For more details, see Tab. 8.

Table 8: Customized configuration of an agent.

Type	Description
agent_type	Whether an agent has ReAct ability.
agent_position	The initial position of an agent.
inventory_capacity	The number of objects an agent can hold.
focal_length	Inversely proportional to viewing angle; shorter focal length corresponding to larger view field.
manipulate_distance	The max distance that an agent's hand can reach.
view_distance	The max distance that an agent's view can reach.

Table 9: Action settings in LangSuit-E.

Action Type	Description
<i>Low-level actions</i>	
move_ahead	Move ahead by 1 step.
turn_left	Turn left by 90 degrees.
turn_right	Turn right by 90 degrees.
pan_left	Move left by 1 step.
pan_right	Move right by 1 step.
pick_up [obj]	Pick up an object identified by [obj].
drop [obj]	Drop the object [obj] to the agent's current position.
put [obj, recep]	Put the object [obj] into the receptacle [recep].
open [obj]	Open object [obj] if it is openable.
close [obj]	close object [obj] if it is closeable.
toggle_on [obj]	Toggle on object [obj] if it is toggleable.
toggle_off [obj]	Toggle off object [obj] if it is toggleable.
slice [obj]	Slice object [obj] if it is sliceable.
stop [answer]	Stop making actions and conclude the final answer in [answer].
<i>High-level actions</i>	
goto [loc]	Go to a location that is in current observation.
heat [obj, heater]	Heat object with a microwave or a stove burner.
cool [obj, cooler]	Cool object with a fridge.
clean [obj, sink]	Clean object with a basin, a sink or a bathtub.
go_explore [room]	Explore around the room.
go_check [recep]	Go to receptacle and check its inside.
go_grab [obj]	Go to object and pick it up.
go_put [recep]	Go to receptacle and put the object in hand into it.
<i>Communicative actions</i>	
chat [message]	Send messages to other agents.
ask [message]	Send queries to users.

F Pseudo-Code for Expert Trajectory Generation

Algorithm 1 shows the pseudo-code for generating expert trajectories as references.

G Examples of 5 prompt strategies for task Instruction Grounding

We show the detailed instruction for different prompt strategies: **Act**, **ReAct**, **ReAct+EmMem**, **Reflexion**, and **Reflexion+EmMem**. We highlight the pivotal instruction sentences.

Algorithm 1: Expert Trajectory Generation

Input: Initialized environment E , start status of environment $\mathcal{S} = \{s_i\}$, target status of environment $\mathcal{T} = \{t_i\}$

Output: a list of actions

- 1 **statusDiff** \leftarrow compare start status and target status by function
- 2 **CompareStatus** (\mathcal{S}, \mathcal{T})
- 3 **plan** \leftarrow obtain a sequence of sub-tasks by SubtaskPlanner (**StatusDiff**)
- 4 **while** **plan** $\neq \emptyset$ **do**
- 5 **subtask** \leftarrow pop a sub-task from **plan**.
- 6 **agentPosition** \leftarrow get current agent position from environment E
- 7 **objectPosition** \leftarrow get object position from subtask
- 8 **moveActions** \leftarrow compute shortest path by AStarPathPlanner (**agentPosition**, **objectPosition**)
- 9 **operationActions** \leftarrow get Operation Actions from subtask
- 10 **for** $action \in$ **moveActions** **do**
- 11 Update **finalActions** by append the $action$ into **finalActions**
- 12 Update environment E by performing $action$
- 13 **for** $action \in$ **operationActions** **do**
- 14 Update **finalActions** by append the $action$ into **finalActions**
- 15 Update environment E by performing $action$
- 16 **return** **finalActions**

G.1 Act

Instruction:

You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see. You can see at most {max_view_steps} step(s) in front

of you, {side_steps} step(s) on your left, {side_steps} step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like `pick_up [object_name]`, `toggle [object_name]` on it.

Inventory: These are the objects you are holding, and you can take an action like `drop` on it.

Feedback: Whether the action is succeed or not and why is it. At each step, you can choose to think your plan or execute an action from one of the following:

``move_ahead``: it means you move ahead by 1 step.

``turn_left``: it means you turn left 90 degrees.

``turn_right``: it means you turn right 90 degrees. ``pick_up [object_name]``:

Use this action to pick up a pickupable and visible object directly in front of you without obstacle. ``drop [object_name]``: it means you places the object in hand at the place just in front of you. ``toggle [object_name]``: it means you toggle a toggleable and visible object directly in front of you without obstacle.

``stop [answer]``: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as ``N/A`` in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with ``Act: ``. For example, ``Act: pick_up [ball]``.

4. You can only operate the object you are facing, if the object is in front of you, on your left or right, take action like `move_ahead`, `turn_left`, `turn_right` to approach it until you are facing it;
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

{example}

Respond YES if you can play this game.

Example:

Task: go to the red box.

Obs: You can see a blue key in front of you; You can see a red box on your right.

Manipulable object: A blue key.

>Act: `turn_right`.

Task:

Task: {task}

Obs: {observation}

What is your next step?

G.2 ReAct

Instruction:

You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see. You can see at most {max_view_steps} step(s) in front of you, {side_steps} step(s) on your left, {side_steps} step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like `pick_up [object_name]`, `toggle [object_name]` on it.

Inventory: These are the objects you are holding, and you can take an action like drop on it.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

``move_ahead``: it means you move ahead by 1 step.

``turn_left``: it means you turn left 90 degrees.

``turn_right``: it means you turn right 90 degrees.

``pick_up [object_name]``: Use this action to pick up a pickupable and visible object directly in front of you without obstacle.

``drop [object_name]``: it means you places the object in hand at the place just in front of you.

``toggle [object_name]``: it means you toggle a toggleable and visible object directly in front of you without obstacle.

``stop [answer]``: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as ``N/A`` in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action or your **thought** that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with ``Act: ``. For example, ``Act: pick_up [cup_0]``.
4. State your **thought** or think process starting with ``Thought: ``
5. You can only operate the object you are facing, if the object is in front of you, on your left or right, take action like `move_ahead`, `turn_left`, `turn_right` to approach

- it until you are facing it;
6. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

{example}

Respond YES if you can play this game.

Example:

Task: go to the red box.

Obs: You can see a blue key in front of you; You can see a red box on your right.

Manipulable object: A blue key.

>Thought: To solve this task, I need to find a red box. There is a blue key just before me and blocked my way. The red box is on my right, so I should turn `right` first to go to the red box.

>Act: `turn_right`.

Task:

Task: {task}

Obs: {observation}

What is your next step? Try to think before act.

G.3 ReAct+EmMem

Instruction:

You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see .You can see at most { max_view_steps} step(s) in front of you, {side_steps} step(s) on your left, {side_steps} step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like `pick_up [`

object_name], toggle [object_name] on it.

Inventory: These are the objects you are holding, and you can take an action like drop on it.

>Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

`move_ahead`: it means you move ahead by 1 step.

`turn_left`: it means you turn left 90 degrees.

`turn_right`: it means you turn right 90 degrees.

`pick_up [object_name]`: Use this action to pick up a pickupable and visible object directly in front of you without obstacle.

`drop [object_name]`: it means you places the object in hand at the place just in front of you.

`toggle [object_name]`: it means you toggle a toggleable and visible object directly in front of you without obstacle.

`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "\N/A\" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action or your **thought** that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with "\Act: \". For example, "\Act: pick_up [cup_0]\".
4. State your **thought** or think process starting with "\Thought: \"
5. You can only operate the object in your sight, if the object is in front of you, on your left or

right, take action like move_ahead, turn_left, turn_right to approach it until you it in your sight;

6. Issue stop action when you think you have achieved the objective.\n7. Remember you are facing NORTH at the begining.

Don't generate anything after stop.

Here are some examples:

{example}

Respond YES if you can play this game.

Example:

Task: go to the red box.

Obs: You can see nothing ahead.

>**Thought:** I see nothing, it could be I have reached the border or I am in a middle grid but there is nothing in front of me. I have taken 0 steps and I am facing NORTH now. I should remember what I saw, so that I can find them easily when necessary; I need to find a red box.

>OK.

>**Act:** move_ahead

>**Feedback:** Action failed. Can not move ahead, because there is an obstacle ahead.

Obs: You see nothing ahead.

>**Thought:** I can't move ahead because of an obstacle but I saw nothing in front of me. so I have reached the border, I have taken 0 step and am facing NORTH now since last action failed. Next I will try to look around.

>OK.

>**Act:** turn_right

>**Feedback:** Action succeeded. Turned right by '90' degrees.

Obs: You see a red box.

>**Thought:** I have reached the border when facing NORTH. I haven taken 2 steps and am facing EAST after turn right at last step. The border is on my leftside. Now I find the red box. Next I need to go to it.

>OK.

>**Act:** move_ahead

>Feedback: Action succeeded. Moved forward by 1 step.
 Obs: You can see a red box.
 >Thought: I am moving along the border now, I have taken 3 steps and am still facing EAST after move ahead at last step. I need to go to the red box.
 >OK.
 >Act: move_ahead()
 >Feedback: Action failed. Can not move ahead, because there is an obstacle ahead.
 Obs: You can see a red box.
 >Thought: I see a red box and can not move ahead, I have taken 3 steps and am still facing EAST since last action failed. I have reached the red box.
 >OK.
 >Act: stop[].

Task:

Task: {task}

Obs: {observation}

What is your next step? Try to summarize your status, recall what you have done and think before act.

G.4 Reflexion

Instruction:

You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see .You can see at most { max_view_steps} step(s) in front of you, {side_steps} step(s) on your left, {side_steps} step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like pick_up [object_name], toggle [object_name]

on it.

Inventory: These are the objects you are holding, and you can take an action like drop on it.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

`move_ahead`: it means you move ahead by 1 step.

`turn_left`: it means you turn left 90 degrees.

`turn_right`: it means you turn right 90 degrees.

`pick_up [object_name]`: Use this action to pick up a pickupable and visible object directly in front of you without obstacle.

`drop [object_name]`: it means you places the object in hand at the place just in front of you.

`toggle [object_name]`: it means you toggle a toggleable and visible object directly in front of you without obstacle.

`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as \"N/A\" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action or your **thought** that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with \"Act: \". For example, \"Act: pick_up [cup_0]\".
4. State your **thought** or think process starting with \"Thought: \"
5. You can only operate the object you are facing, if the object is in front of you, on your left or right, take action like move_ahead,

turn_left, turn_right to approach it until you are facing it;
6. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

{example}

Respond YES if you can play this game.

Example:

Task: go to the red box.

Obs: You can see a blue key in front of you; You can see a red box on your right.

Manipulable object: A blue key.

>**Thought:** To solve this task, I need to find a red box. There is a blue key just before me and blocked my way. The red box is on my right, so I should turn_right first to go to the red box.

>**Act:** turn_right.

Task:

Task: {task}

Your memory from last trails is: \{memory\}

Obs: {observation}

What is your next step? Try to think before act.

G.5 Reflexion+EmMem

Instruction:

You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see. You can see at most {max_view_steps} step(s) in front of you, {side_steps} step(s) on your left, {side_steps} step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like pick_up [object_name], toggle [object_name] on it.

Inventory: These are the objects you are holding, and you can take an action like drop on it.

>**Feedback:** Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

`move_ahead`: it means you move ahead by 1 step.

`turn_left`: it means you turn left 90 degrees.

`turn_right`: it means you turn right 90 degrees.

`pick_up [object_name]`: Use this action to pick up a pickupable and visible object directly in front of you without obstacle.

`drop [object_name]`: it means you places the object in hand at the place just in front of you.

`toggle [object_name]`: it means you toggle a toggleable and visible object directly in front of you without obstacle.

`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as \"N/A\" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action or your **thought** that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with **Act:** \". For example, **Act:** pick_up [cup_0]\".
4. State your **thought** or think process starting with **Thought:** \"

5. You can only operate the object in your sight, if the object is in front of you, on your left or right, take action like `move_ahead`, `turn_left`, `turn_right` to approach it until you see it in your sight;
 6. Issue stop action when you think you have achieved the objective.
 7. Remember you are facing NORTH at the beginning.
- Don't generate anything after stop.

Here are some examples:

{example}

Respond YES if you can play this game.

Example:

Task: go to the red box.

Obs: You can see nothing ahead.

>**Thought:** I see nothing, it could be I have reached the border or I am in a middle grid but there is nothing in front of me. I have taken 0 steps and I am facing NORTH now. I should remember what I saw, so that I can find them easily when necessary; I need to find a red box.

>OK.

>**Act:** `move_ahead`

>**Feedback:** Action failed. Can not move ahead, because there is an obstacle ahead.

Obs: You see nothing ahead.

>**Thought:** I can't move ahead because of an obstacle but I saw nothing in front of me. so I have reached the border, I have taken 0 step and am facing NORTH now since last action failed. Next I will try to look around.

>OK.

>**Act:** `turn_right`

>**Feedback:** Action succeeded. Turned right by '90' degrees.

Obs: You see a red box.

>**Thought:** I have reached the border when facing NORTH. I haven taken 2 steps and am facing EAST after turn right at last step. The border is on my leftside. Now I find the red box. Next I need to

go to it.

>OK.

>**Act:** `move_ahead`

>**Feedback:** Action succeeded. Moved forward by 1 step.

Obs: You can see a red box.

>**Thought:** I am moving along the border now, I have taken 3 steps and am still facing EAST after move ahead at last step. I need to go to the red box.

>OK.

>**Act:** `move_ahead()`

>**Feedback:** Action failed. Can not move ahead, because there is an obstacle ahead.

Obs: You can see a red box.

>**Thought:** I see a red box and can not move ahead, I have taken 3 steps and am still facing EAST since last action failed. I have reached the red box.

>OK.

>**Act:** `stop[]`.

Task:

Task: {task}

Your memory from last trails is: {memory}

Obs: {observation}

What is your next step? Try to summarize your status, recall what you have done and think before act.

H Examples of different task prompts

H.1 Instruction Grounding

System: You are an autonomous intelligent agent tasked with completing a minigrid game. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see. You can see at most 7 step(s) in front of you, 3 step(s) on your

left, 3 step(s) on your right.

Manipulable object: The object that you are facing and you can take an operate action like pick_up [object_name], toggle [object_name] on it.

Inventory: These are the objects you are holding, and you can take an action like drop on it.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

``move_ahead``: it means you move ahead by 1 step.

``turn_left``: it means you turn left 90 degrees.

``turn_right``: it means you turn right 90 degrees.

``pick_up [object_name]``: Use this action to pick up a pickupable and visible object directly in front of you without obstacle.

``drop [object_name]``: it means you places the object in hand at the place just in front of you.

``toggle [object_name]``: it means you toggle a toggleable and visible object directly in front of you without obstacle.

``stop [answer]``: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue a action or your **thought** that is valid given the current observation
2. You should only issue ONE action at a time, and ONLY action is enough.
3. Generate the action in the correct format starting with "**Act**: ". For example, "**Act**: pick_up [cup_0]".
4. State your **thought** or think process starting with "**Thought**: "

5. You can only operate the object you are facing, if the object is in front of you, on your left or right, take action like move_ahead, turn_left, turn_right to approach it until you are facing it;

6. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

Task: go to the red box.

Obs: You can see a blue key in front of you; You can see a red box on your right.

Manipulable object: A blue key.

>**Thought:** To solve this task, I need to find a red box. There is a blue key just before me and blocked my way. The red box is on my right, so I should turn_right first to go to the red box.

>**Act:** turn_right.

Respond YES if you can play this game.

Assistant: YES

System: **Task:** pick up the red key

Obs: You can see a yellow door, a yellow key in front of you; You can see a green door on your left; You can see a green box, a red key on your right.

What is your next step?

Assistant: **Thought:** To accomplish the task, I need to pick up the red key. The red key is on my right, so the first step is to turn right.

System: OK.

Assistant: **Act:** turn_right.

System: **Feedback:** Action succeeded. Turned right by '90' degrees.

Obs: You can see a red key in front of you; You can see a green door, a blue key, a green box, a yellow key on your left; You can see a purple ball, a purple door on your right.

Assistant: **Thought:** Now that I'm facing the red key, I can pick it up to accomplish my task.

System: OK.

Assistant: **Act:** pick_up [red key].

System: Feedback: Action failed.
Failed to pick up red key. You can only pickup the object one step in front of you without obstacle.

Obs: You can see a red key in front of you; You can see a green door,a blue key,a green box,a yellow key on your left; You can see a purple ball,a purple door on your right.

Assistant: **Thought:** The red key is still in front of me, but I may need to move closer to pick it up.

System: OK.

Assistant: **Act:** move_ahead.

System: Feedback: Action succeeded.
Moved forward by 1 step.

Obs: You can see a red key in front of you; You can see a green door,a blue key,a green box on your left; You can see a purple ball on your right.

Assistant: **Thought:** The red key is now directly in front of me. I can pick it up now.

System: OK.

Assistant: **Act:** pick_up [red key].

System: Feedback: Action failed.
Failed to pick up red key. You can only pickup the object one step in front of you without obstacle.

Obs: You can see a red key in front of you; You can see a green door,a blue key,a green box on your left; You can see a purple ball on your right.

Assistant: **Thought:** I must have been too far to pick up the red key. I should move one step ahead to reach the key.

System: OK.

Assistant: **Act:** move_ahead.

System: Feedback: Action succeeded.
Moved forward by 1 step.

Obs: You can see a red key in front of you; You can see a green door,a blue key on your left;
Manipulable object: a red key.

Assistant: **Thought:** I am now facing the red key and can pick it up.

System: OK.

Assistant: **Act:** pick_up [red key].

System: Feedback: Action succeeded.
You picked a red key up.

Inventory: You are now holding a red key.

Obs: You can see a green door,a blue key on your left.

System: [SUCCESS] You have completed the task. Congratulations!

System: DONE!

H.2 Rearrangement

System: You are an autonomous intelligent agent tasked with rearranging a virtual home. **System** will randomly change between 1 to 5 objects in the room. Your goal is to identify which objects have changed and reset those objects to their original state.

Here are the information you will have :

Original status: The original room status that you should remember, pay your attention to the position and openness states of each of the objects. Then find out all the changed objects and reset them to their original states.

Obs: These are the objects you can see . You can see at most 8.0 step(s) in front of you; 60 degrees on your left, 60 degrees on your right.

Inventory: These are the objects you are holding. You can hold at most 1 objects.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

- `move_ahead`: it means you move ahead by 1 step.
- `turn_left`: it means you turn left 90 degrees.
- `turn_right`: it means you turn right 90 degrees.
- `pick_up [object_name]`: Use this action to pick up a pickupable and visible object within 8.0 step(s).

`drop [object_name]`: it means you place the object in hand at the place just in front of you.
`open [object_name]`: it means you open an openable and visible object within 8.0 step(s).
`close [object_name]`: it means you close a closeable and visible object.
`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as 'N/A' in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue ONE action or ONE **thought** that is valid given the current observation
2. When you issue a action, and ONLY ONE action is enough, and then wait for your feedback from system.
3. Generate the action in the correct format starting with "**Act:** " For example, "**Act:** pick_up [cup_0]" Do not add adjectives before object name.
4. State your **thought** or think process starting with "**Thought:** "
5. You can ONLY operate the object within your view distance.
6. If got action failed feedback, try to take other actions to proceed the task.
7. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

Original status: In the north of the room, there is a armchair_0; a pen_1; In the east of the room, there is a chair_0;

Obs: You can see nothing ahead.

> **Thought:** To solve the task, I need to explore the room to find out what objects have been changed. I

see a pen_1, because I am in the south of the room, so the pen_1 is in the south of the room, but I remember the pen_1 should be in the north of the room according to the original status. So I need to pick it up and put it back.

OK.

> **Act:** move_ahead

Respond YES if you can play this game.

Rearrange Agent: YES

System: Let's start.

Original state: In the north of the room, there is a cabinet_2; a cabinet_3; a cabinet_13; a cabinet_16; a cabinet_19; a cabinet_22; a microwave_0; a sink_0; a sink_1, in/on it you can see a dish sponge_0. In the northeast of the room, there is a lightswitch_0. In the east of the room, there is a cabinet_7; a cabinet_20; a countertop_0, in/on it you can see a coffeemachine_0, a fork_0, a papertowelroll_0, a peppershaker_0, a saltshaker_0, a spatula_0, a spoon_0; a drawer_0; a drawer_1; a garbagecan_0; a stoveknob_2. In the southeast of the room, there is a cabinet_17; a countertop_1, in/on it you can see a cup_0, a plate_0; a drawer_2; a stoveburner_0; a stoveburner_1; a stoveburner_2, in/on it you can see a pot_0; a stoveburner_3; a stoveknob_0; a stoveknob_1; a stoveknob_3. In the south of the room, there is a diningtable_0, in/on it you can see a apple_0, a bread_0, a lettuce_0, a mug_0, a potato_0, a tomato_0; a stool_0; a window_0. In the southwest of the room, there is a cabinet_5; a cabinet_12; a cabinet_14; a cabinet_21. In the west of the room, there is a cabinet_0; a cabinet_4; a cabinet_9; a cabinet_11; a fridge_0, in/on it you can see a egg_0. In the northwest of the room, there is a cabinet_1; a cabinet_6; a

cabinet_8; a cabinet_10; a cabinet_15; a cabinet_18; a countertop_2, in/on it you can see a bowl_0, a butterknife_0, a faucet_0, a faucet_1, a knife_0, a pan_0, a soapbottle_0, a toaster_0; a drawer_3.

Obs: In front of you, You see a cabinet_0; a cabinet_4; a cabinet_9; a cabinet_11; an opened fridge_0, there is a egg_0 in/on it. On your left, you see a cabinet_2; a cabinet_10; a cabinet_13; a cabinet_18; a countertop_2; a bowl_0; a faucet_1; a knife_0; a pan_0; a toaster_0; a diningtable_0; a apple_0; a bread_0; a butterknife_0; a dish sponge_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a spoon_0; a tomato_0; a drawer_3; a microwave_0; a sink_0; a sink_1; a stool_0; a window_0. On your right, you see a cabinet_3; a cabinet_22; a faucet_0.

what is your next step?

Assistant: Thought: As there are many objects in the room, I need to identify which objects have changed. Since I can see an egg in the fridge but the fridge is open, I will close the fridge to reset it to its original state.

Assistant: Act: close fridge_0.

System: Feedback: Action succeeded.

You closed fridge_0.

Obs: In front of you, You see a cabinet_0; a cabinet_4; a cabinet_9; a cabinet_11; a fridge_0. On your left, you see a cabinet_2; a cabinet_10; a cabinet_13; a cabinet_18; a countertop_2; a bowl_0; a faucet_1; a knife_0; a pan_0; a toaster_0; a diningtable_0; a apple_0; a bread_0; a butterknife_0; a dish sponge_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a spoon_0; a tomato_0; a drawer_3; a microwave_0; a sink_0; a sink_1; a

stool_0; a window_0. On your right, you see a cabinet_3; a cabinet_22; a faucet_0.

Assistant: Thought: Now that the fridge is closed, I can see the original status of the room. I need to check if there are any changes in the room.

Assistant: Act: turn_left.

System: Feedback: Action succeeded. Turned left by '90' degrees.

Obs: In front of you, You see a diningtable_0; a apple_0; a dish sponge_0; a spoon_0; a window_0. On your left, you see a bread_0; a butterknife_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a tomato_0; a stool_0; a stoveburner_0; a pot_0; a stoveburner_3. On your right, you see a cabinet_17; a countertop_1; a cup_0; a plate_0; a drawer_2; a fridge_0; a stoveburner_1; a stoveburner_2; a stoveknob_0; a stoveknob_3.

Assistant: Thought: I can see a dish sponge in the dining table which was not there before. So, I need to pick it up and put it back to its original position.

Assistant: Act: pick_up dish sponge_0.

System: Feedback: Action succeeded.

You picked dish sponge_0 up.

Inventory: dish sponge_0.

Obs: In front of you, You see a diningtable_0; a apple_0; a spoon_0; a window_0. On your left, you see a bread_0; a butterknife_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a tomato_0; a stool_0; a stoveburner_0; a pot_0; a stoveburner_3. On your right, you see a cabinet_17; a countertop_1; a cup_0; a plate_0; a drawer_2; a fridge_0; a stoveburner_1; a stoveburner_2; a stoveknob_0; a stoveknob_3.

Assistant: Thought: I need to put the dish sponge back to the sink_0 where it was originally placed.

Assistant: **Act:** move_ahead.

System: **Feedback:** Action succeeded.

Moved forward by 1 step.

Obs: In front of you, You see a diningtable_0; a window_0. On your left, you see a apple_0; a bread_0; a lettuce_0; a spoon_0; a stoveburner_0; a pot_0; a stoveburner_1; a stoveburner_2; a stoveknob_0. On your right, you see a cabinet_17; a countertop_1; a cup_0; a plate_0; a butterknife_0; a potato_0; a saltshaker_0; a soapbottle_0; a tomato_0; a drawer_2; a stool_0.

Assistant: **Thought:** The sink_0 is not in my view. I need to turn to my right to find it.

Assistant: **Act:** turn_right.

System: **Feedback:** Action succeeded.

Turned right by '90' degrees.

Obs: In front of you, You see a cabinet_0; a cabinet_9; a cabinet_11; a fridge_0. On your left, you see a cabinet_2; a cabinet_4; a cabinet_10; a cabinet_13; a countertop_2; a bowl_0; a faucet_1; a knife_0; a pan_0; a toaster_0; a diningtable_0; a apple_0; a bread_0; a butterknife_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a spoon_0; a tomato_0; a drawer_3; a microwave_0; a sink_0; a sink_1; a stool_0; a window_0. On your right, you see a cabinet_3; a cabinet_5; a cabinet_12; a cabinet_21; a cabinet_22; a faucet_0.

Assistant: **Thought:** The sink_0 is in front of me. I need to move ahead and put the dish sponge back to the sink_0.

Assistant: **Act:** move_ahead.

System: **Feedback:** Action succeeded.

Moved forward by 1 step.

Obs: In front of you, You see a cabinet_0; a cabinet_11; a fridge_0. On your left, you see a cabinet_1; a cabinet_2; a cabinet_4; a cabinet_6; a

cabinet_9; a cabinet_10; a cabinet_18; a countertop_2; a bowl_0; a faucet_1; a knife_0; a pan_0; a toaster_0; a diningtable_0; a apple_0; a bread_0; a butterknife_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a spoon_0; a tomato_0; a drawer_3; a stool_0; a window_0. On your right, you see a cabinet_3; a cabinet_5; a cabinet_12; a cabinet_13; a cabinet_14; a cabinet_21; a faucet_0; a microwave_0; a sink_0; a sink_1.

Assistant: **Thought:** I am in front of the sink_0. I need to drop the dish sponge_0 to put it back to its original position.

Assistant: **Act:** drop dish sponge_0 in sink_0.

System: **Feedback:** Action succeeded.

You dropped dish sponge_0.

Inventory: Empty

Obs: In front of you, You see a cabinet_0; a cabinet_11; a fridge_0; a dish sponge_0. On your left, you see a cabinet_1; a cabinet_2; a cabinet_4; a cabinet_6; a cabinet_9; a cabinet_10; a cabinet_18; a countertop_2; a bowl_0; a faucet_1; a knife_0; a pan_0; a toaster_0; a diningtable_0; a apple_0; a bread_0; a butterknife_0; a lettuce_0; a potato_0; a saltshaker_0; a soapbottle_0; a spoon_0; a tomato_0; a drawer_3; a stool_0; a window_0. On your right, you see a cabinet_3; a cabinet_5; a cabinet_12; a cabinet_13; a cabinet_14; a cabinet_21; a faucet_0; a microwave_0; a sink_0; a sink_1.

Assistant: **Thought:** I have successfully reset the changed objects to their original states. Now, there is no change in the room. I can stop here.

Assistant: **Act:** stop []

System: You stopped the game.

System: {'misplaced': 0.0, 'fixed': 1.0}

System: [SUCCESS] You have completed the task. Congratulations!

System: DONE!

H.3 Interactive QA

System: Imagine you are a robot that can move and see and open objects in a 2D grid kitchen environment. At the beginning of this task I will give you a question about the environment and all the objects in front of you within your view size:

1. **Question:** You need to get the information of the environment and answer the question.
2. **Obs:** Visible objects list

At each step you get some information from me, and you respond one robot **thought** and one robot action each time, nothing else, and then wait for feedback. You will get these information from me, remember these information are all from me, you can only generate Robot action:

1. **Obs:** Visible objects list
2. **Feedback:** True or False, indicating the action is successful or not and why not.

What you can do:
wait for me to give you next round feedback including Scene and Successful action and something else.

Remember carefully, after you output robot **thought** and robot action, you should wait for me to give you feedback.

Robot **thought**:
thought [**thought** info]: it means your **thought** about how to get the answer.

Robot action:
move_ahead: it means robot move ahead by 1 step.
turn_left: it means robot turn left 90 degrees.

turn_right: it means robot turn right 90 degrees.

open [object_name]: it means robot open an openable object, open [fridge_0] means open fridge_0, and remember do not issue action open if you have opened the object.

answer [answer_info]: do this to answer the question.

Generate the **thought** and open and answer action in the correct format. Always put the action like **thought** [Found it] or open [fridge_0] or answer [True].

Remember carefully, you generate **thought** and action, then wait for my feedback. And there is a space after the action name.

Now I will give you 3 examples:

1. **Question:** Is there a mug in the room?

Obs: In front of you, You see a Basketball_1, a Spoon_1, a Stool_1; On your left, you see a baseball_1.

>**thought** [I should try my best to find the mug in the room.]

>move_ahead

Feedback: Action succeeded.

Obs: In front of you, You see a Basketball_2, a Spoon_1, a Plate_1.

>**thought** [Keep searching the room.]

>turn_right

Feedback: Action succeeded.

Obs: In front of you, You see a Basketball_3, a Pot_1, a fridge_1.

Feedback: Action succeeded.

Obs: In front of you, you see a mug_1, a Pot_1, a fridge_1.

>**thought** [Found it!]

>answer [True]

2. **Question:** I think there is a potato in the fridge. Is that correct?

Obs: In front of you, You see a potato_0, a winebottle_0, a cabinet_2; On your left, you see a baseball_1.

```

>thought [I should try my best to find
    the potato in the room.]
>move_ahead
Feedback: Action succeeded.
Obs: In front of you, You see a
    potato_0;a winebottle_0;a
    cabinet_2;On your left, you see a
    fridge_0.
>thought [Keep searching the room.]
>turn_right
Feedback: Action succeeded.
Obs: In front of you, You see a
    fridge_0.
>thought [Maybe in the fridge?]
>open [fridge_0]
Feedback: Action succeeded.
Obs: In/on it you You see mug_1,
    potato_0 in/on it.
>thought [Found it!]
>answer [True]

3.Question: Count the number of apples
    in this room.
Obs: In front of you, You see a bowl_0
    ; a stool_0; a tomato_0; On your
    left, you see a butterknife_0; a
    cabinet_6; a coffeemachine_0; a
    book_0;
>thought [I should try my best to find
    all the apples in the room.]
>move_ahead
Feedback: Action succeeded.
Obs: In front of you, You see a bowl_0
    ; a stool_0; On your left, you see
    a butterknife_0; a cabinet_6; a
    coffeemachine_0;
>thought [zero, keep looking.]
>move_ahead
Feedback: Action failed.
Obs: In front of you, You see a bowl_0
    ; a stool_0; On your left, you see
    a butterknife_0; a cabinet_6; a
    coffeemachine_0;
>thought [zero, keep looking.]
>turn_right
Feedback: Action succeeded.
Obs: In front of you, You see a
    butterknife_0; a cabinet_6; a
    coffeemachine_0;
>thought [zero, keep looking.]
>move_ahead
Feedback: Action succeeded.

```

```

Obs: In front of you, You see a
    apple_0; a potato_1;a fridge_0; On
    your left, you see a cabinet_2;
>thought [One, keep looking.]
>open [fridge_0]
Feedback: Action succeeded.
Obs: In/on it you see apple_0;
    potato_1; apple_1; On your left,
    you see a cabinet_2;
>thought [Two, keep looking.]
>move_ahead
Feedback: Action succeeded.
Obs: In front of you, You see a
    potato_1; On your left, you see a
    stool_2;
>thought [Two, keep looking.]
>move_ahead
Feedback: Action succeeded.
Obs: In front of you, You see a
    apple_3; On your left, you see a
    stool_2;
>thought [Three!]
>answer [3]

```

```

Remember carefully,you need to
    generate noth thought and action!
    Then wait for my feedback,for
    example,what you need to generate
    is: thought [Maybe in the fridge?]
open [fridge_0].What i give you is :
    Feedback: Action succeeded.
Obs: In/on it you You see mug_1,
    potato_0 in/on it.
Respond YES if you can play this game.
Assistant: YES
System: Question: Do we have any cups?
Obs: You see nothing. You can try to
    take action like move_ahead,
    turn_left or turn_right to explore
    the room.
Assistant: Act: move_ahead
System: Feedback: Action succeeded.
    Moved forward by '0.25' meter(s).
Obs: You see nothing. You can try to
    take action like move_ahead,
    turn_left or turn_right to explore
    the room.
Assistant: move_ahead
System: Feedback: Action succeeded.
    Moved forward by '0.25' meter(s).

```

Obs: You see nothing. You can try to take action like `move_ahead`, `turn_left` or `turn_right` to explore the room.

Assistant: Act: `move_ahead`

System: Feedback: Action succeeded. Moved forward by '0.25' meter(s).

Obs: You see nothing. You can try to take action like `move_ahead`, `turn_left` or `turn_right` to explore the room.

Assistant: Act: `turn_right`

System: Feedback: Action succeeded. Turn right by '90' degrees.

Obs: On your left, you see a `lightswitch_0`; On your right, you see a `bottle_1`, a `creditcard_1`, a `mug_1`, a `creditcard_1`, a `stool_1`,

Assistant: Act: `move_ahead`

System: Feedback: Action succeeded. Moved forward by '0.25' meter(s).

Obs: On your left, you see a `lightswitch_0`; On your right, you see a `bottle_1`, a `creditcard_1`, a `stool_1`,

Assistant: Act: `turn_right`

System: Feedback: Action succeeded. Turn right by '90' degrees.

Obs: In front of you, You see a `stoveknob_0`; a `stoveknob_1`; a `stoveknob_2`; a `cabinet_5`; a `stoveknob_3`; a `houseplant_1`; a `spoon_1`; On your left, you see a `bowl_5`; a `drawer_1`; a `drawer_2`; a `cabinet_0`; a `cabinet_1`; a `window_0`; a `sink_0`; a `drawer_6`; a `cabinet_6`; a `drawer_8`; a `toaster_1`; a `winebottle_1`; a `vase_3`; a `coffeemachine_1`; a `bowl_5`; a `lettuce_1`; a `stool_1`; a `faucet_0`; On your right, you see a `bowl_3`, a `bottle_1`, a `bottle_1`, a `knife_1`, a `soapbottle_2`, a `vase_1`, a `bowl_3`, a `creditcard_1`, a `pot_1`, a `papertowelroll_1`, a `fridge_0`, a `lightswitch_0`, a `mug_1`, a `creditcard_1`,

Assistant: Thought: Found it!

Assistant: Act: `answer [True]`

System: Your are right!

System: DONE!

H.4 Household

System: You are an autonomous intelligent agent tasked with navigating a virtual home. You will be given a household task. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see .

Inventory: These are the objects you are holding.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

``move_ahead``: it means you move ahead by 1 step.

``turn_left``: it means you turn left 90 degrees.

``turn_right``: it means you turn right 90 degrees.

``pick_up [object_name]``: Use this action to pick_up a pick_upable and visible object within your manipulation distance.

``drop [object_name]``: it means you places the object in hand at the place just in front of it.

``open [object_name]``: it means you open an openable and visible object within your manipulation distance.

``close [object_name]``: it means you close a closeable and visible object.

``toggle_on [object_name]``: it means you toggle a toggleable and visible object on.

``toggle_off [object_name]``: it means you toggle a toggleable and visible object off.

``slice [object_name]``: it means you slice a sliceable and visible object.

`put [object_name, receptacle_name]`: it means you places the object in hand into/onto the receptacle.
`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue ONE action or ONE **thought** that is valid given the current observation
2. When you issue a action, and ONLY ONE action is enough, and then wait for your feedback.
3. Generate the action in the correct format starting with "Act: ". For example, "Act: pick_up [cup_0]".
4. State your **thought** or think process starting with "Thought: "
5. You can ONLY operate the object within your view distance.
6. If got action failed feedback, try to take other actions to proceed the task.
7. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.

Here are some examples:

Task: put a clean lettuce in diningtable.

Obs: You can see nothing ahead.

> **Thought**: To solve the task, I need to find and take a lettuce, then clean it with sinkbasin, then put it in diningtable.

OK.

> **Act**: move_ahead

Respond YES if you can play this game.

Alfred Agent: YES

System: Task: Examine a phone by the light of the lamp in the corner.

Obs: In front of you, You see a cellphone_0. On your left, you see a houseplant_0. On your right, you see a drawer_2; a sidetable_0;

a lamp_2; a shelf_6; a shelf_7; a shelf_9; a shelf_11.

What is your next step? Let's think before act.

Assistant: **Thought**: To complete the task, I need to turn on the lamp, then pick up the phone to examine it. The lamp is within sight, so I can start by turning it on.

System: OK.

Assistant: **Act**: toggle_on [lamp_2]

System: **Feedback**: Action succeeded. You toggled lamp_2 on.

Obs: In front of you, You see a cellphone_0. On your left, you see a houseplant_0. On your right, you see a drawer_2; a sidetable_0; a lamp_2; a shelf_6; a shelf_7; a shelf_9; a shelf_11.

Assistant: **Act**: pick_up [cellphone_0]

System: **Feedback**: Action succeeded. You picked cellphone_0 up.

Inventory: cellphone_0.

Obs: On your left, you see a houseplant_0. On your right, you see a drawer_2; a sidetable_0; a lamp_2; a shelf_6; a shelf_7; a shelf_9; a shelf_11.

Assistant: **Act**: pick_up [cellphone_0]

System: **Feedback**: Action failed. There is no object "CellPhone |-05.53|+01.04|+00.17" existing. Please operate the object in sight.

Obs: On your left, you see a houseplant_0. On your right, you see a drawer_2; a sidetable_0; a lamp_2; a shelf_6; a shelf_7; a shelf_9; a shelf_11.

Assistant: **Thought**: I have picked up the cellphone and it is in my inventory. I'm now examining it by the light of the lamp. The task is complete.

System: OK.

Assistant: **Act**: stop [examined the phone by the light of the lamp]

System: You stopped the game.

System: 1.0

System: [SUCCESS] You have completed the task. Congratulations!

System: DONE!

H.5 Multi-agent Teach

System: You are commander, an autonomous intelligent agent tasked with navigating a virtual home. You will be given a household task. These tasks will be accomplished through specific actions you post and through collaboration with follower. Note that you cannot interact with objects. You can only view the scene and guide follower to complete the task by using `chat`. You can not know the action history of follower.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see. You can see at most 40.0 step(s) in front of you; 60 degrees on your left, 60 degrees on your right.

Inventory: These are the objects you are holding. You can hold at most 0 objects.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

`move_ahead`: it means you move ahead by 1 step.
`move_back`: it means you move back by 1 step.
`turn_left`: it means you turn left 90 degrees.
`turn_right`: it means you turn right 90 degrees.
`no_op`: it means you do nothing.
`pan_left`: it means you move left by 1 step.
`pan_right`: it means you move right by 1 step.
`open_progress_check`: it means you check the progress of the task, you will be told which condition

is failed.

`chat [chat_message]`: it means you send a chat to follower to guide it to do the task.
`select_oid [object_name]`: it means that you can select one object and get its position relative to the follower.
`search_object [object_type]`: it means that you can search for an object type and select one, which will then get its position relative to the follower.
`go_to [object_name]`: it means you travel to an available location next to the object.
`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue an action or your **thought** that is valid given the current observation
2. You should only issue one action at a time.
3. Generate the action in the correct format. For example, pick_up [cup].
4. State your **thought** or think process starting with "**Thought:** "
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.
6. Please do not generate the same operation as the last one.

Here are some examples:

Task: put a clean lettuce on diningtable.

Obs: You can see nothing ahead.

> **Thought:** I need to check the task progress.

> **Act:** open_progress_check

Feedback: Action failed. One clean lettuce needs to be on diningtable.

> **Thought**: I need to tell the task to follower.
> **Act**: chat [Please put a clean lettuce in diningtable.]
Feedback: Action succeeded.
follower: Where can I find lettuce?
Thought: To solve the task, I need to guide follower to find and take a lettuce, then clean it with sinkbasin, then put it in diningtable.
> **Act**: select_oid [lettuce_0]
Feedback: Action succeeded. lettuce_0 is in front and left of follower.

Respond YES. if you can play this game
.

System: You are follower, an autonomous intelligent agent tasked with navigating a virtual home. You will be given a household task. These tasks will be accomplished through specific actions you post and through collaboration with commander. Note that you can interact with objects but have no access to task information. You can use `chat` to ask commander for information to accomplish the task. You can not know the action history of follower.

Here's the information you'll have:

Task: This is the task you are trying to accomplish.

Obs: These are the objects you can see . You can see at most 8.0 step(s) in front of you; 60 degrees on your left, 60 degrees on your right.

Inventory: These are the objects you are holding. You can hold at most 1 objects.

Feedback: Whether the action is succeed or not and why is it.

At each step, you can choose to think your plan or execute an action from one of the following:

`move_ahead`: it means you move ahead by 1 step.

`move_back`: it means you move back by 1 step.

`turn_left`: it means you turn left 90 degrees.

`turn_right`: it means you turn right 90 degrees.

`no_op`: it means you do nothing.

`pan_left`: it means you move left by 1 step.

`pan_right`: it means you move right by 1 step.

`open_progress_check`: it means you check the progress of the task, you will be told which condition is failed.

`chat [chat_message]`: it means you send a chat to ask for information from commander to complete the task.

`go_to [object_name]`: it means you travel to an available location next to the object.

`pick_up [object_name]`: Use this action to pick_up a pickupable and visible object within 8.0 step(s).

`place [receptacle_name]`: it means you places the object in hand at the receptacle.

`open [object_name]`: it means you open an openable and visible object within 8.0 step(s).

`close [object_name]`: it means you close a closeable and visible object within 1 step ahead.

`toggle_on [object_name]`: it means you toggle a toggleable and visible object on within 1 step ahead.

`toggle_off [object_name]`: it means you toggle a toggleable and visible object off within 1 step ahead.

`slice [object_name]`: it means you slice a sliceable and visible object within 1 step ahead.

`stop [answer]`: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

- To be successful, it is very important to follow the following rules:
1. You should only issue an action or your **thought** that is valid given the current observation
 2. You should only issue one action at a time.
 3. Generate the action in the correct format. For example, pick_up [cup].
 4. State your **thought** or think process starting with "**Thought**: "
 5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.
 6. Please do not generate the same operation as the last one.

Here are some examples:

Task: You can use `chat` to ask commander for task information.
Obs: You can see nothing ahead.
> **Thought**: I need to wait information from commander.
> **Act**: no_op
Feedback: Action succeeded.
commander: Please put a clean lettuce in diningtable.
> **Thought**: I need to ask commander for the location of the lettuce.
> **Act**: chat [Where can I find lettuce ?]
Feedback: Action succeeded.
Obs: You can see nothing ahead.
> **Thought**: To solve the task, I need to find and take a lettuce, then clean it with sinkbasin, then put it in diningtable.
But I don't know where lettuce is, I can wait information from follower or explore by myself.
> **Act**: pan_left
Feedback: Action succeeded.
Obs: In front of you, You see a lettuce_0.

Respond YES. if you can play this game
commander: YES.

System: **Task**: Put all Pillow on any ArmChair.

Obs: In front of you, You see a lightswitch_0; a closed drawer_0; a sidetable_0, there is a deskclamp_1, a creditcard_0, a remotecontrol_0, on it. On your left, you see a shelf_0; a shelf_1; a shelf_2, there is a deskclamp_1, on it; a shelf_3. On your right, you see a garbagecan_0; a painting_0; a dresser_0, there is a pillow_0, a deskclamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it; a closed drawer_3; a closed cabinet_3.

What is your next step? Let's think and act.

follower: YES.

System: **Task**: You can use `chat` to ask commander for task information.

Obs: On your left, you see a dresser_0, there is a pillow_0, a deskclamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it. On your right, you see a closed drawer_1; a closed cabinet_0.

What is your next step? Let's think and act.

commander: **Thought**: I need to find an ArmChair to place the pillows on. But first, I'll ask the follower to pick up the pillow from the dresser.

commander: **Act**: chat [Please pick up the pillow from the dresser.]

System: **Feedback**: Action succeeded.

follower: **Thought**: The pillow is visible on the dresser. I need to go to the dresser and pick up the pillow.

follower: **Act**: go_to [dresser_0]

System: **Feedback**: Action succeeded. Go to dresser_0. In front of you, You see a garbagecan_0; a closed drawer_2; a closed cabinet_1. On your left, you see a dresser_0,

there is a pillow_0, a desklamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it; a closed drawer_3; a closed cabinet_3. On your right, you see a closed drawer_1; a closed cabinet_0.

commander: **Thought**: Next, I need to guide the follower to an armchair to put the pillow on it. I should find an armchair first.

commander: **Act**: search_object [ArmChair]

System: Feedback: Action succeeded. armchair_0 is at the left rear of follower.

follower: **Act**: pick_up [pillow_0]

System: Feedback: Action succeeded. You picked pillow_0 up.

Inventory: pillow_0.

Obs: In front of you, You see a garbagecan_0; a closed drawer_2; a closed cabinet_1. On your left, you see a dresser_0, there is a pillow_0, a desklamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it; a closed drawer_3; a closed cabinet_3. On your right, you see a closed drawer_1; a closed cabinet_0.

commander: **Act**: chat [please place the pillow on the armchair to your left rear.]

System: Feedback: Action succeeded.

follower: **Thought**: Before placing the pillow on the armchair, I should ensure the armchair is visible. I will turn left twice to make the armchair visible.

follower: **Act**: turn_left

System: Feedback: Action succeeded. Turn left by '90' degrees.

Obs: You see nothing. You can try to take action like move_ahead, turn_left or turn_right to explore the room.

commander: **Act**: search_object [pillow]

System: Feedback: Action succeeded. pillow_0 is in front and left of

follower.

follower: **Act**: turn_left

System: Feedback: Action succeeded. Turn left by '90' degrees.

Obs: In front of you, You see a armchair_0, there is a keychain_0, on it; a closed drawer_2; a closed cabinet_1; a closed drawer_4. On your left, you see a closed cabinet_2; a dresser_0, there is a pillow_0, a desklamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it.

commander: **Act**: chat [please pick up the pillow in front of you.]

System: Feedback: Action succeeded.

follower: **Act**: go_to [armchair_0]

System: Feedback: Action succeeded. Go to armchair_0. In front of you, You see a armchair_0, there is a keychain_0, on it; a closed drawer_2; a closed cabinet_1; a closed drawer_4. On your left, you see a closed cabinet_2; a dresser_0, there is a pillow_0, a desklamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it.

commander: **Act**: chat [please place the pillow on the same armchair.]

System: Feedback: Action succeeded.

follower: **Act**: place [armchair_0]

System: Feedback: Action succeeded. You put pillow_0 to armchair_0.

Obs: In front of you, You see a armchair_0, there is a keychain_0, a pillow_0, on it; a closed drawer_2; a closed cabinet_1; a closed drawer_4. On your left, you see a closed cabinet_2; a dresser_0, there is a pillow_0, a desklamp_2, a newspaper_1, a keychain_1, a cellphone_0, an opened laptop_0, it's empty, a vase_0, on it.

System: [SUCCESS] You have completed the task. Congratulations!

System: DONE!

H.6 Multi-agent Cooperation

System: You are an autonomous intelligent agent tasked with navigating a virtual home. Your name is Alice. You are hurry to finish a housework with your friend Bob together. There are four kinds of rooms, livingroom, kitchen, bedroom, bathroom. You will be given a household task. These tasks will be accomplished through the use of specific actions you and your friend can issue. You have two hands to pickup, hold, put and check objects.

Here's the information you will have:

Task: This is the task you and your friend Bob are trying to accomplish.

At each step, you can choose to think your plan or issue an action from one of the following:

- ``chat [message].``: You can use this action to send a message to your friend Bob.
- ``go_explore [room].``: Use this action to go to a room and explore around.
- ``go_check [container].``: Use this action to go to a unchecked container and check its inside. You must have at least one free hands to check.
- ``go_grab [object].``: Use this action to go to an object and grab it.
- ``go_put [container].``: Use this action to go to the container and place the object in the container. You need to make sure you have the object(s) you need for the task.
- ``stop [answer].``: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue an action or your **thought** that is valid given the current observation
2. You should only issue one action at a time.
3. Generate the action in the correct format starting with "**Act:** ". For example, "**Act:** go_grab [phone_0].".
4. State your **thought** or think process starting with "**Thought:** "
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.
6. You should generate chats like this : chat [Hello, how can I help you ?], taking care to enclose them in square brackets.

Respond YES. if you can play this game

System: You are an autonomous intelligent agent tasked with navigating a virtual home. Your name is Bob. You are hurry to finish a housework with your friend Alice together. There are four kinds of rooms, livingroom, kitchen, bedroom, bathroom. You will be given a household task. These tasks will be accomplished through the use of specific actions you and your friend can issue. You have two hands to pickup, hold, put and check objects.

Here's the information you will have:

Task: This is the task you and your friend Alice are trying to accomplish.

At each step, you can choose to think your plan or issue an action from one of the following:

- ``chat [message].``: You can use this action to send a message to your friend Alice.
- ``go_explore [room].``: Use this action to go to a room and explore around.

``go_check [container].``: Use this action to go to a unchecked container and check its inside. You must have at least one free hands to check.

``go_grab [object].``: Use this action to go to an object and grab it.

``go_put [container].``: Use this action to go to the container and place the object in the container. You need to make sure you have the object(s) you need for the task.

``stop [answer].``: Issue this action when you believe the task is complete. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue an action or your **thought** that is valid given the current observation
2. You should only issue one action at a time.
3. Generate the action in the correct format starting with "**Act:** ". For example, "**Act:** go_grab [phone_0].".
4. State your **thought** or think process starting with "**Thought:** "
5. Issue stop action when you think you have achieved the objective. Don't generate anything after stop.
6. You should generate chats like this : chat [Hello, how can I help you ?], taking care to enclose them in square brackets.

Respond YES. if you can play this game .

Alice: YES.

System: **Task:** Find and put 1 wine onto the coffeetable_0.

Obs: You have already found and put pudding_0, juice_0, juice_1 onto the coffeetable_0. You are holding nothing. You are in the kitchen, where you found unchecked containers kitchencabinet_0,

kitchencabinet_1, kitchencabinet_2, kitchencabinet_3, kitchencabinet_4, kitchencabinet_5, kitchencabinet_6, kitchencabinet_7, stove_0, dishwasher_0, fridge_0, fridge_1, microwave_0. You don't know where Bob is. The livingroom is unexplored. The bedroom is unexplored. The bathroom is unexplored.

What is your next step? Let's think and act.

Bob: YES.

System: **Task:** Find and put 1 wine onto the coffeetable_0.

Obs: You have already found and put pudding_0, juice_0, juice_1 onto the coffeetable_0. You are holding nothing. You are in the bedroom, where you found an unchecked container cabinet_0. You don't know where Alice is. The livingroom is unexplored. The kitchen is unexplored. The bathroom is unexplored.

What is your next step? Let's think and act.

Alice: **Thought:** The task is to find and put 1 wine onto the coffeetable_0. I am currently in the kitchen with several unchecked containers. I will start by checking these containers as the wine could be in any of them.

Alice: **Act:** go_check [kitchencabinet_0].

System: **Feedback:** Action succeeded. You opened kitchencabinet_0.

Obs: In it you see nothing

Bob: **Thought:** Since I need to find the wine and I am currently in an unexplored bedroom, I'll first check the unchecked container in the bedroom, the cabinet_0. If the wine is not there, I'll move to the next unexplored room.

Bob: **Act:** go_check [cabinet_0].

System: **Feedback:** Action succeeded. You opened cabinet_0.

Obs: In it you see book_3

Alice: **Thought**: The first kitchen cabinet was empty. I'll check the next one.

Alice: **Act**: go_check [kitchencabinet_1]

System: Feedback: Action succeeded. You opened kitchencabinet_1.

Obs: In it you see nothing

Bob: **Thought**: The wine is not in the cabinet_0 in the bedroom, so I need to explore other rooms. The kitchen is a good place to start as it is common to store wine there.

Bob: **Act**: go_explore [kitchen]

System: Feedback: Action succeeded. Go to kitchen_0. You have already found and put pudding_0, juice_0, juice_1 onto the coffeetable_0. You are holding nothing. You are in the kitchen, where you found unchecked containers kitchencabinet_2, kitchencabinet_3, kitchencabinet_4, kitchencabinet_5, kitchencabinet_6, kitchencabinet_7, stove_0, dishwasher_0, fridge_0, fridge_1, microwave_0. You also see Alice here in the kitchen, she is holding nothing. The livingroom is unexplored. You found nothing in the bedroom. The bathroom is unexplored.

Alice: **Act**: go_check [kitchencabinet_2]

System: Feedback: Action succeeded. You opened kitchencabinet_2.

Obs: In it you see book_1

Bob: **Act**: go_check [fridge_0]

System: Feedback: Action succeeded. You opened fridge_0.

Obs: In it you see wine_0

Alice: **Act**: go_check [kitchencabinet_3]

System: Feedback: Action succeeded. You opened kitchencabinet_3.

Obs: In it you see nothing

Bob: **Act**: go_grab [wine_0]

System: Feedback: Action succeeded. You picked wine_0 up.

Obs: You have already found and put pudding_0, juice_0, juice_1 onto

the coffeetable_0. You are holding wine_0. You are in the kitchen, where you found unchecked containers kitchencabinet_4, kitchencabinet_5, kitchencabinet_6, kitchencabinet_7, stove_0, dishwasher_0, fridge_1, microwave_0. You also see Alice here in the kitchen, she is holding nothing. The livingroom is unexplored. You found nothing in the bedroom. The bathroom is unexplored.

Alice: **Act**: go_check [kitchencabinet_4]

System: Feedback: Action succeeded. You opened kitchencabinet_4.

Obs: In it you see nothing

Bob: **Act**: go_explore [livingroom]

System: Feedback: Action succeeded. Go to livingroom_0. You have already found and put pudding_0, juice_0, juice_1 onto the coffeetable_0. You are holding wine_0. You are in the livingroom, where you found coffeetable_0. Last time you saw Alice was in the kitchen, she was holding nothing. You found unchecked containers kitchencabinet_5, kitchencabinet_6, kitchencabinet_7, stove_0, dishwasher_0, fridge_1, microwave_0 in the kitchen. You found nothing in the bedroom. The bathroom is unexplored.

Alice: **Act**: go_check [kitchencabinet_5]

System: Feedback: Action succeeded. You opened kitchencabinet_5.

Obs: In it you see book_2

Bob: **Act**: go_put [coffeetable_0]

System: Feedback: Action succeeded. You put wine_0 on coffeetable_0.

Obs: You have already found and put pudding_0, juice_0, juice_1, wine_0 to the coffeetable_0. You are holding nothing. You are in the livingroom, where you found coffeetable_0. Last time you saw Alice was in the kitchen, she was holding nothing. You found unchecked containers

kitchencabinet_5, kitchencabinet_6,
kitchencabinet_7, stove_0,
dishwasher_0, fridge_1,
microwave_0 in the kitchen. You
found nothing in the bedroom. The
bathroom is unexplored.

System: [SUCCESS] You have completed
the task. Congratulations!

System: DONE!