
Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents

Zihao Wang^{1,2}, Shaofei Cai^{1,2}, Guanzhou Chen³, Anji Liu⁴, Xiaojian Ma^{4*}, Yitao Liang^{1,5*}
Team CraftJarvis

¹Institute for Artificial Intelligence, Peking University

²School of Intelligence Science and Technology, Peking University

³School of Computer Science, Beijing University of Posts and Telecommunications

⁴Computer Science Department, University of California, Los Angeles

⁵Beijing Institute for General Artificial Intelligence (BIGAI)

{zhwang, caishaofei}@stu.pku.edu.cn, rayment@bupt.edu.cn
liuanji@cs.ucla.edu, xiaojian.ma@ucla.edu, yitaol@pku.edu.cn

Abstract

We investigate the challenge of task planning for multi-task embodied agents in open-world environments.² Two main difficulties are identified: 1) executing plans in an open-world environment (e.g., Minecraft) necessitates accurate and multi-step reasoning due to the long-term nature of tasks, and 2) as vanilla planners do not consider how easy the current agent can achieve a given sub-task when ordering parallel sub-goals within a complicated plan, the resulting plan could be inefficient or even infeasible. To this end, we propose “Describe, Explain, Plan and Select” (**DEPS**), an interactive planning approach based on Large Language Models (LLMs). DEPS facilitates better error correction on initial LLM-generated *plan* by integrating *description* of the plan execution process and providing self-explanation of feedback when encountering failures during the extended planning phases. Furthermore, it includes a goal *selector*, which is a trainable module that ranks parallel candidate sub-goals based on the estimated steps of completion, consequently refining the initial plan. Our experiments mark the milestone of the first zero-shot multi-task agent that can robustly accomplish 70+ Minecraft tasks and nearly double the overall performances. Further testing reveals our method’s general effectiveness in popularly adopted non-open-ended domains as well (i.e., ALFWorld and tabletop manipulation). The ablation and exploratory studies detail how our design beats the counterparts and provide a promising update on the ObtainDiamond grand challenge with our approach. The code is released at <https://github.com/CraftJarvis/MC-Planner>.

1 Introduction

Developing multi-task agents that can accomplish a vast and diverse suite of tasks in complex domains has been viewed as one of the key milestones towards generally capable artificial intelligence [34, 1, 5, 10, 25]. To enable such capabilities, earlier works have suggested employing a hierarchical goal execution architecture [2, 4], where a planner generates action plans that would then be executed by low-level goal-conditioned controllers. This architecture has been delivering promising progress in

*Corresponding Author.

²We borrow the term “open world” from the game community. It highlights that the agent can navigate inside a diverse environment and accomplish open-ended tasks freely.

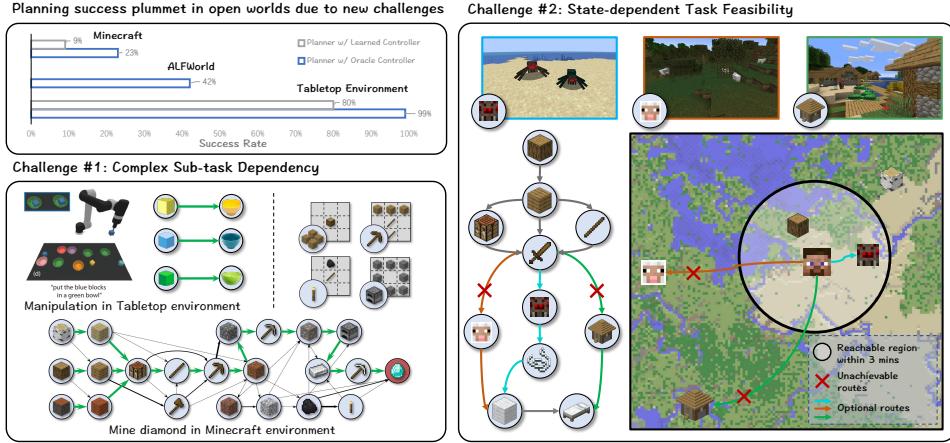


Figure 1: **Planning success rates plummet in open worlds due to new challenges.**

many robotics domains, including table-top and mobile manipulation [42, 4], 2D shape drawing [20] and table rearrangement [17]. However, whether such success can be transferred to a more open-ended world with unlimited exploration areas and internet-scale knowledge remains open [14, 10, 13, 12, 19].

To understand the gap, we run Inner Monologue [17], a general and competitive hierarchical goal execution model on a typical open-world domain Minecraft [18, 14, 10] and two classical robotic environments ALFWorld [38] and Tabletop environments [37, 4]. The algorithm uses a Large Language Model (LLM) based planner that contains domain-specific knowledge for all three environments. In all environments, we use either an Oracle goal-conditioned controller or a learned one. Results are shown in the bar plot in Figure 1. First, even when the Oracle controller is used, the success rate of executing Minecraft tasks is much less than that of the other environments. Next, the task failure rate becomes even higher in Minecraft when the learned controller is substituted. Both failures originate from unique challenges brought by open-world environments, which we identify in the following.

First, compared to canonical environments (e.g., Atari [28] and robotic control suite [37]), open worlds have highly abundant object types with complex dependency and relation. As a result, ground-truth plans typically involve a long sequence of sub-goals with strict dependencies. As Figure 1 challenge #1 suggests, it requires at least 13 sub-goals executed in proper order to obtain a diamond in Minecraft, while in Tabletop a task is typically no more than a few consecutive sub-goals.

Another challenge brought by the complicated tasks in an open-ended world is the feasibility of the produced plans. Consider the example shown in Figure 1 (challenge #2). To craft a bed in Minecraft, the fastest way is by either slaughtering a sheep to obtain wool, which can be used to craft beds, or collecting beds from a village. However, since no sheep or village is reachable by the agent within 3 minutes of gameplay, to craft a bed efficiently, the agent should choose to slaughter a spider and use materials (e.g., string) it drops to craft wool, and then a bed. That is, when dealing with a task that can be completed by executing multiple possible sequences of sub-goals, the planner should be able to select the best route based on the current state of the agent. However, the complex and diverse state distribution of open-world environments makes state awareness hard to achieve.

To tackle these problems, we propose “Describe, Explain, Plan and Select” (**DEPS**), an interactive planning approach based on Large Language Models (LLMs) to alleviate the aforementioned issues. The key to tackling the first challenge is to effectively adjust the generated plan upon failure. Specifically, whenever the controller fails to complete a sub-goal, a *descriptor* will summarize the current situation as text and send it back to the LLM-based planner. We then prompt the LLM as an *explainer* to locate the errors in the previous plan. Finally, a *planner* will refine the plan using information from the descriptor and explainer. To improve the feasibility of generated plans conditioned on the current state, which is the second identified challenge, we use a learned goal-*selector* to choose the most accessible sub-task based on the proximity to each candidate sub-goal.

Our experiments are conducted on 71 tasks in open-ended Minecraft without any demonstration. Given the goal-conditioned controller for atom sub-tasks (i.e., mine log and mine stone), our zero-

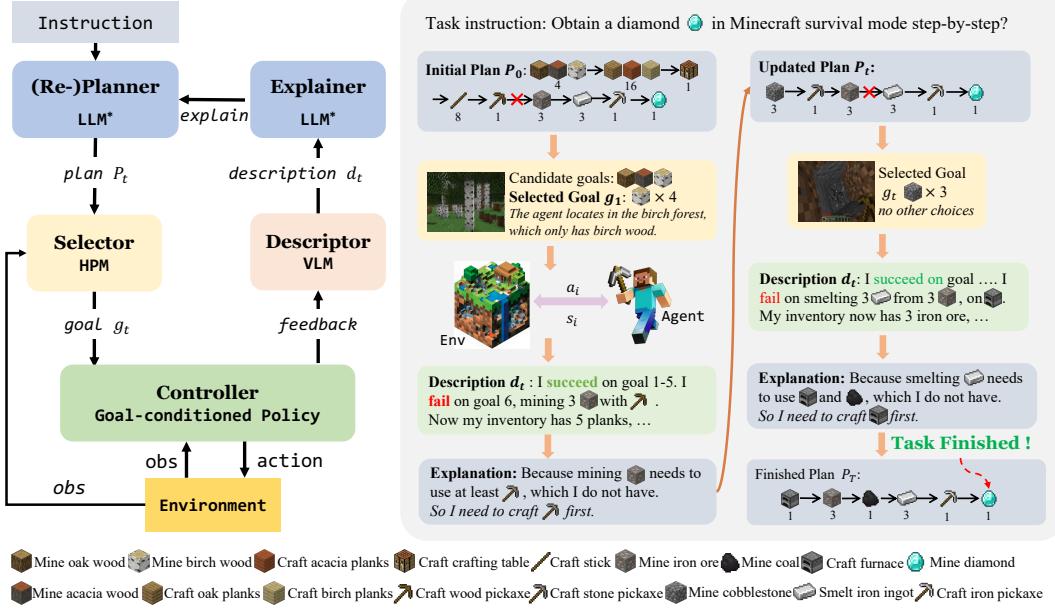


Figure 2: Overview of our proposed interactive planner architecture.

shot³ LLM-based planner can finish all tasks within a limited horizon (3000-12000 steps for different tasks). We find DEPS outperforms all language planner baselines by nearly doubling the overall success rate, with the same initial state and goal-conditioned controller. Our ablation and exploratory studies then explain how our approach beats the counterparts and becomes the first planning-based agent that accomplishes the challenging ObtainDiamond task. DEPS does not require any planning training for the environment. Additionally, DEPS achieves between on-par and more than 50% relative improvement over existing or concurrent LLM-based planning methods on non-open-ended robotics domains such as ALFWORLD [38] and Tabletop environments [37].

2 Background

We aim to develop an agent capable of solving long-horizon goal-reaching tasks using image observations and language goals. To accomplish this, we propose a combined approach involving goal-conditioned policies (termed controllers) and a planner. The goal-conditioned policies are trained to complete sub-goals, while the planner decomposes long-horizon tasks into a series of K short-horizon sub-goals, g_1, \dots, g_K , to be executed by the controller. At each time step t , the goal-conditioned policy $\pi(a_t | s_t, g_k)$ generates an action a_t based on the current state s_t and the specified sub-goal g_k .

Planning with Large Language Models Previous works have shown that LLMs such as InstructGPT [30] and Codex [8] can be used as zero-shot planners to generate sub-goal sequences for various tasks in embodied environments [16, 39]. Formally, given the task description T as prompt p , LLM acts as a planner to decode T into K sub-goals, g_1, \dots, g_K , which are then executed one by one by the low-level controller $\pi(a_t | s_t, g_k)$ to accomplish the task.

However, the above pipeline suffers from both challenges identified in Section 1. Regarding the first challenge, the probability of generating a flawless plan directly from the task description decreases significantly as the required number of sub-goals increases. Moreover, even when the LLM generates a correct plan, it is very likely that the plan is highly inefficient given the agent’s current state (challenge #2). Prior works mostly focus on solving the first challenge by providing environmental feedback to the LLM through affordance functions [4], success detector [20] or scene descriptor [17]. However, although these approaches work well on many non-open-ended domains, they still suffer from high failure rates in open-world environments.

³Similar to [5, 16], “zero-shot” here means no gradient updates are performed. However we provide some related demonstrations as prompts during inference time.

3 Towards Reliable Planning in Embodied Open-World Environments

In this section, we first give an overview of our proposed interactive planning framework “Describe, Explain, Plan, and Select” (DEPS) for solving complex and long-horizon tasks in open-world environments (Sec. 3.1). Next, in Section 3.2, we elaborate how DEPS iteratively refines its plan to combat the first identified challenge. Section 3.3 introduces the *selector* module that is used to identify efficient plans in response to the second identified challenge.

3.1 DEPS Overview

As demonstrated in Figure 2, our agent (DEPS) consists of an event-triggered Descriptor, a Large Language Model (LLM) as Explainer and Planner, a goal Selector based on horizon prediction and a goal-conditioned controller. In the following, we use Minecraft as a running example to better elaborate our agent. Note that DEPS can be directly applied to other (non-)open-ended tasks.

We take a large language model (LLM) as a zero-shot *planner* of the agent to complete tasks. Given a goal command (e.g., ObtainDiamond) as task T , the LLM-based planner decomposes this high-level task into a sequence of sub-goals $\{g_1, \dots, g_K\}$, as the initial plan P_0 . The goals are instructions in natural language, such as mine oak wood (in Minecraft), find two cups (in ALFWORLD), put block A on top of block B (in Tabletop Manipulation).

As described in Section 2, a controller is then invoked to execute the provided sub-goals sequentially through a goal-conditioned policy $\pi(a | s, g)$. However, the initial plan provided by the planner often contains errors, which results in execution failures of the controller. For example, the goal can not be finished only with a wooden pickaxe as shown in Figure 2. When failure pops up, the *descriptor* will summarize the current state s_t and execution outcome of the most recent goal into text d_t and send it to the LLM. The LLM will first try to locate the errors in the previous plan P_{t-1} by *self-explanation*, e.g., the goal need to be executed with a stone pickaxe . Then it will re-plan the current task T and generate a revised plan P_t according to the explanation. In this process, the LLM is also treated as an *explainer* in addition to the *planner* role. The Descriptor, Explainer, and Planner will be detailed in Section 3.2.

$$\begin{aligned}
 \text{Description} : d_t &= f_{\text{DESC}}(s_{t-1}), \\
 \text{Explanation} : e_t &= f_{\text{EX}}(d_t), \\
 \text{Prompt} : p_t &= \text{CONCAT}(p_{t-1}, d_t, e_t), \\
 \text{Plan} : P_t &= f_{\text{LM}}(p_t), \\
 \text{Goal} : g_t &\sim f_{\text{S}}(P_t, s_{t-1}), \\
 \text{Action} : a_t &\sim \pi(a_t | s_{t-1}, g_t)
 \end{aligned} \tag{1}$$

As shown in Equation (1), DEPS will iteratively update the plan P_t until the task is finished, where f_{DESC} is the descriptor model, f_{LM} denotes the language model as explainer and planner, f_{S} is the selector model, π is goal-conditioned policies from the controller.

To filter out inefficient plans, the *selector* is trained to predict the number of time steps remaining to achieve every goal g_k in a set of parallel goals given the current state s_t . When the generated plan contains alternative routes, the selector uses this information to choose a suitable goal as the current goal g_t . For example, the horizon predicted by the selector of goal acacia tree is less than goal oak tree in Savanna biome, which leads to chop acacia tree as current goal g_t .

3.2 Describe, Explain and Plan with LLM Generates Executable Plans

Current LLM-based planners usually query the LLM once at the beginning of every episode and use the output plan throughout the episode [16, 39]. However, as demonstrated by Figure 1, such one-shot planning methods often fail on long-horizon tasks that require many sub-goals. This is caused by two major issues. First, since the correct plan for long-horizon tasks needs to respect various complex preconditions, it is extremely hard for the LLM to generate a flawless plan directly from the task instructions, resulting in failure when simply following the initial plan. Additionally, due to the unpredictable transition dynamics, some incidents may happen during the execution and make the initial plan non-executable. To remedy these problems, existing methods introduce feedback (e.g.,

Prompt 1 Planner prompt template, Python-like code

```
def craft_wooden_axe(initial_inventory={}):
    # step 1: mine 3 logs
    mine(obj = {"log":3}, tool = None)
    # step 2: craft 12 planks from 3 logs
    craft(obj = {"planks":12}, materials = {"log":3}, tool = None)
    # step 3: craft 4 sticks from 2 planks
    craft(obj = {"stick":4}, materials = {"planks":2}, tool = None)
    # step 4: craft 1 crafting_table from 4 planks
    craft(obj = {"crafting_table":1}, materials = {"planks":4}, tool = None)
    # step 5: craft 1 wooden_axe from 3 planks and 2 sticks on crafting table
    craft(obj = {"wooden_axe":1}, {"planks": 3, "stick": 2}, tool = "crafting_table")
    return "wooden_axe"
```

from success detector or scene descriptor) to reflect on the results of previous executions [17, 20, 4]. However, merely informing the LLM whether a sub-goal is completed is often insufficient to correct the planning error.

To remedy this, we propose “describe, explain and plan”, a new interactive planning method to generate more executable and explainable plans. We start with rewriting the prompt into an interactive dialogue format as in ChatGPT [30] so that subsequent feedback can be passed to the LLM effectively. The produced plan is also augmented with the preconditions and effects of each goal. The structured prompt improves the readability and interpretability of the plan and facilitates error-locating when the execution fails later, as demonstrated in Prompt 1.

The *descriptor* will then collect the feedback generated by the agent during the execution of the task. The feedback can be practically obtained either by a person (human feedback [4]), or by a pre-trained vision-language model CLIP [33]. While the previous type of feedback needs intensive human involvement, the latter from the pre-trained model needs to be fine-tuned for the specific domain, which decreases the automation and generalization of the agent. On the contrary, Minecraft returns the ‘info’ and other high-level observations (such as biome, GPS, and compass), we can easily translate the unstructured information into structured language. Therefore we take the symbolic information available in the game and translate it into feedback description d_t in this work. To avoid carrying unrelated information in the prompt, we further distill plan-related messages (e.g., inventory information, biome) as final event-level description d_t as demonstrated in Figure 2.

Notably, we also treat the LLM as an *explainer* to explain why the previous plans P_{t-1} failed. Specifically, by analyzing the current state from description d_t and precondition of current goal g_t , the explainer can identify the reason why the current goal cannot be executed successfully. As shown in Figure 2, the reason may be *the current goal requires the use of an iron pickaxe, but the tool is not prepared in advance, or the current goal requires the use of 3 planks, but the currently available planks are not enough*. To implement this, we provide few-shot demonstrations to the LLM as in chain-of-thoughts prompting [41], as shown in Prompt 1. Finally, the LLM goes back to its role as a *planner* and re-plans the task with the explicit explanation of existing bugs in the previous plan P_{t-1} , ultimately generating an updated plan P_t according to the explanation.

3.3 Horizon-Predictive Selector Yields Efficient Plans

Due to the abundance of objects and the compositional nature of their functionalities, there often exist multiple feasible plans to complete a task, i.e., there are usually multiple paths for the completion of a particular goal. However, despite the feasibility of all such plans, most of them are highly inefficient to execute in the current episode. For example, as shown in Figure 2, obtaining a wood can be done by chopping oak trees , birch trees , or acacia trees . But only oak trees are available in the plains biome. So the planner needs to choose oak trees since it is more efficient, as the agent does not need to travel to another biome.

On the other hand, there is no strict sequential requirement for some goals in the plan P_t , i.e., $g_i, g_j \sim P_t$ enjoy the same precondition, which means g_i and g_j can be executed in any order. As shown in Figure 1, the choice of different paths (sequences) may affect the execution efficiency of the plan P_t as one goal might be closer to the agent. Always choosing the closer goal to execute first could yield more efficient plans and improve the final success rate under a limited episode length. Moreover, the dynamic nature of open-world environments further amplifies the impact of efficient



Figure 3: **Selection Demonstration from “Selector”.** Given parallel sub-goals, i.e. candidate skills, our Selector will determine the sequence in which to carry out these sub-goals based on their current proximity to the agent and modify the original plan produced by the LM planner.

plans on the success rate. For example, in Minecraft, if the agent chooses to execute a further goal like `collect wood` first, the much closer target `sheep` may disappear and be hard to find again.

In order to improve the efficiency of our plans, we propose to use a *selector* that selects the most efficient path with the highest execution success rate as the final plan. Specifically, we design a state-aware selector to choose the nearest goal under state s_t as the current goal g_t from the candidate goal sets in plan P_t . It predicts the goal distribution $p(g_t|s_t, P_t)$ under the current state s_t and plan P_t , where $g_t \in G_t$, G_t describes all current executable goals in P_t . A straight way to implement the selector is to leverage the semantic similarity between the current state and the goal text using a vision-language model (VLM) such as CLIP [33]. Nevertheless, this may not exactly reflect the difficulty of completing the goal since VLM lacks practical experience. For example, an “oak tree” in front of the agent could lead to high semantic similarity for the “chopping tree” goal, but it may be inefficient to achieve this goal if a canyon is in the middle between the agent and the oak tree.

To mitigate this, we implement a horizon-predictive selector that embeds practical task experience to accurately rank the goals based on their efficiency and feasibility. Here, we define the horizon of a goal $h_t(g) := T_g - t$ as the remaining time steps to complete the given goal, where T_g is the time of completing goal g . This metric accurately reflects how quickly we can achieve the given goal from the current state. To estimate the horizon, we learn a neural network μ to fit the offline trajectories by minimizing the entropy loss $-\log \mu(h_t(g) | s_t, g)$, where h_t is the ground-truth horizon in trajectories of completing goal g . Therefore, the goal distribution can be formulated as follows:

$$f(g_t | s_t, P_t) = \frac{\exp(-\mu(g_t, s_t))}{\sum_{g \in G_t} \exp(-\mu(g, s_t))}. \quad (2)$$

We set goal-sensitive Impala CNN [6] as the backbone of the selector. In practice, the horizon predictive selector can be jointly trained with the controller policies and share the backbone parameters [6].

4 Experiments

This section analyzes and evaluates our proposed “describe, explain, plan, and select” (DEPS) method. To minimize performance variation caused by the low-level controller, we standardize all experiments with one controller learned by behavior cloning. We refer to the details of this controller in Appendix ???. In Section 4.1, we introduce our testing environments and our evaluation task set, consisting of the hardest 71 tasks from MCU SkillForgeChain [22]. In Section 4.2, we report our performance in the context of existing LLM-based planners. Ablation studies are conducted in Section 4.3. Finally, we pay close attention to the hardest task, `ObtainDiamond`, which is long-hailed as a major challenge in the community. The experiments on ALFWorld and Tabletop Manipulation environments are shown in Appendix ??.

4.1 Experimental Setup

Environment and Task Setting We first evaluate our proposed method in Minecraft, a popular open-world environment with both challenges discussed in Section 1. For better reflecting the performance of DEPS, we choose three Minecraft environments with different versions for better

evaluation, including Minedojo [10] with Minecraft 1.11.2, MineRL [3] with Minecraft 1.16.5, and MC-TextWorld [22] with Minecraft 1.19.2. Rules and items have something different in the above three Minecraft environments, which can better evaluate the dynamic and interactive planning abilities of DEPS.

Table 1: Attributes of 8 meta tasks covering Task101: We evaluate the algorithm on Minecraft Task101. We group the consisted 71 task into 8 different meta groups, with each focusing on testing a different aspect of our proposed method.

Meta	Name	Number	Example Task	Max. Steps	Initial Inventory	Given Tool
MT1	Basic	14	Make a wooden door.	3000	Empty	Axe
MT2	Tool (Simple)	12	Make a stone pickaxe.	3000	Empty	Axe
MT3	Hunt and Food	7	Cook the beef.	6000	Empty	Axe
MT4	Dig-Down	6	Mine coal.	3000	Empty	Axe
MT5	Equipment	9	Equip the leather helmet.	6000	Empty	Axe
MT6	Tool (Complex)	7	Make shears and bucket.	6000	Empty	Axe
MT7	IronStage	13	Obtain an iron sword.	6000	Empty	Axe
MT8	Challenge	1	Obtain a diamond!	12000	Empty	Axe

We choose 71 tasks from the Minecraft Universe Benchmark SkillForgeChain [22] for evaluation. These tasks are related to items that can be obtained in the Minecraft overworld. To better present the results, we divide the 71 Minecraft tasks into 8 meta groups according to the ingredients and function of the tasks, i.e., MT1-MT8. The instruction for every task is written in natural language, e.g., make a wooden door in MT1 (Basic group) and obtain a diamond in MT8 (Challenge group), as illustrated in Table 1. Considering how long it typically takes human players to complete each task as a ballpark [14], we set different maximum episode steps for different meta tasks from 3000 (for easiest **Basic** tasks) to 12000 (for the hardest **Challenge** tasks). The names, number of required skills, and functions of all tasks are listed in Appendix ???. We give an empty inventory for every task in Survival mode and require the agent to obtain every item from the environment by itself. Note that our agent will be summoned in different environments randomly for each evaluation. Biomes and initial positions are also different each time. Following the previous work [18], we take the success rate as the evaluation metric.

Baselines We compare DEPS with other language-based planners, including GPT as Zero-shot Planner(GPT) [16], ProgPrompt(PP) [39], Chain-of-Thought(CoT) [41], Inner Monologue(IM) [17], and Code as Policies(CaP) [20]. For all baseline models, we use the same demonstration example in the prompt, the same LM model from OpenAI, and the same controller in all tasks for a fair comparison. Since these methods were not originally experimented with Minecraft, we reproduce them to conform to the Minecraft specification based on prompt and feedback template design. All planner methods access the LLM model through OpenAI API (text-davinci-03 model [30] for GPT, CoT, and IM, and code-davinci-02 model [8] for PP, CaP, and Ours). All hyper-parameters of LLM (including the *temperature* and *best_of*, etc.) are kept as default. We also list the full prompt of all different methods in Appendix ??.

4.2 Main Results

Every task is executed 30 times and the average results in Minedojo [10] for every meta task are listed in Table 2. Our approach achieves the best performance with all meta tasks. As the complexity of the task increases from MT1-MT8, the planner usually needs to give more accurate task steps (i.e., longer goal sequence) to achieve the final task. Therefore the success rate of all agents decreases with the reasoning steps increasing. Starting from MT6, almost all existing LLM-based planners fail (nearly 0 success rate). DEP (w/o Selector) already consistently beats existing LLM-based planners in all meta tasks with a significant margin. This validates that “describe, explain and plan” can estimate the reason for current plan failure and correct the original flawed plans. Due to the limited maximum episode length and restricted control success rate for a hard goal (e.g., Mine diamond with iron_pickaxe), the final success rate is still capped.

Table 2: Success rates of DEPS and existing LLM planners on Minecraft Task101. The full task-by-task list is in Appendix ??.

Methods	MT1	MT2	MT3	MT4	MT5	MT6	MT7	MT8	AVG
GPT[16, 30]	25.85±24.8	47.88±31.5	10.78±14.6	7.14±9.0	1.98±5.9	0.0±0.0	0.0±0.0	0.0±0.0	15.42
PP[39]	30.61±23.6	40.09±30.6	17.13±19.1	16.00±17.3	3.21±4.9	0.47±1.3	0.60±2.2	0.0±0.0	16.88
CoT[41]	40.24±30.8	55.21±26.8	6.82±11.6	4.76±8.2	1.73±5.2	0.0±0.0	0.0±0.0	0.0±0.0	18.89
IM[17]	46.89±31.4	53.73±20.8	3.64±6.9	18.41±17.4	4.57±7.4	0.64±1.7	1.02±2.5	0.0±0.0	21.64
CaP[20]	60.08±17.3	60.11±20.24	8.72±9.7	20.33±21.0	2.84±4.6	0.63±1.3	0.60±2.2	0.0±0.0	25.77
DEP	75.70±10.4	66.13±13.4	45.69±16.2	43.35±20.2	15.93±13.9	5.71±3.7	4.60±7.1	0.50±0.5	39.36
DEPS	79.77±8.5	79.46±10.6	62.40±17.9	53.32±29.3	29.24±27.3	13.80±8.0	12.56±13.3	0.59±0.5	48.56

In addition, *selector* also greatly improves the final task success rate of the agent (from **DEP w/o Selector to DEPS**). Hard meta tasks usually require the completion of multiple sub-goals (up to dozens of goals), thus bringing more flexibility and providing more candidate goals for the Selector. At the same time, as the agent conducts experiments with limited episode length, it also places high demands on the efficiency of the plan. Therefore, the Selector brings a significant improvement on efficiency-sensitive tasks such as MT7 (up to **+2.7** times success rate).

Robustness on different controller and different Minecraft versions We also evaluate DEPS on MineRL [3] and MC-Textworld [22]. Note that DEPS is a planning method, which needs to equip the goal-conditioned controller for interacting with the Minecraft environments. We choose MC-Controller [6] and Steve-1 [21] as controllers to interact with Minedojo and MineRL, respectively. These two methods are all control policies that perceive visual partial observations and produce mouse and keyboard actions. While MC-Textworld is a text world, which only keeps the Minecraft crafting recipes and mining rules. So MC-Textworld does not require the controller. The DEPS results of the task set MT1-MT8 on different Minecraft environments are shown in Table 3. The results report that DEPS can generate effective plans in various Minecraft environments. The results on MC-Textworld [22] also show that the performance drops on more difficult task sets from MT6 to MT8 are mainly from the controller limitation.

Table 3: Success rates of DEPS under different Minecraft environments.

Environment	Version	Controller	MT1	MT2	MT3	MT4	MT5	MT6	MT7	MT8
MineDojo [10]	1.11.2	[6]	79.77	79.46	62.40	53.32	29.24	13.80	12.56	0.59
MineRL [3]	1.16.5	[21]	84.05	80.32	24.25	36.21	9.16	17.22	16.79	1.84
MC-Textworld [22]	1.19.2	-	100.00	90.00	80.00	56.25	64.71	57.14	69.57	50.00

4.3 Ablation Study

We conduct ablation experiments to investigate the number of candidate executable goals for different Selector models and the specific impact of the rounds of DEPS.

4.3.1 Ablation on Selector

We verify the robustness of our proposed Selector under different parallel goals. The agent is asked to complete 2, 3, and 4 candidate goals (the precondition is consistent for all goals), respectively. The goals of the task correspond to different kinds of mobs or materials.

We report the final success rate of our method (DEP) with different selector implementations, including using a fixed sequence of goals, a random sequence of goals, and selecting a goal based on MineCLIP [10], CLIP [33], and our horizon-predictive Selector (HPS). As Figure 4 shows, in one round of parallel candidate goals, an improvement of $\Delta=+22.3\%$, $+29.2\%$, $+32.6\%$ is obtained using our horizon-predictive Selector compared to not any selector (i.e., fixed plan), respectively.

At a limited episode length, e.g., 1000 steps, goal-model shows a greater advantage, which proves that goal-model can improve the execution efficiency of the plan in embodied environments. In addition, compared to using vision-language models such as CLIP [33] and MineCLIP [10] as a goal model, horizon-predictive has the best performance due to better estimation of the horizon information. The curve trend also demonstrates that agents with Selector scale up under large amounts of goals in an open-world environment.

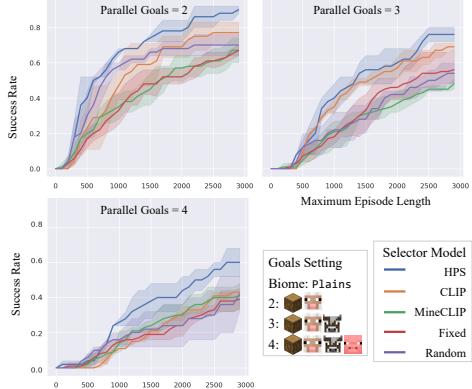


Figure 4: The success rates of DEPS with different selectors under varying numbers of parallel goals and maximum episode lengths.

Table 4: Success Rate of DEPS under different maximum rounds of re-planning. Round 0 represents the vanilla Planner w/o the re-planning process. ∞ represents the re-planning process will not end until task success or reaching the maximum horizon, which is still limited by the maximum tokens of LLMs. The maximum number of rounds for Codex is around 7-8 rounds.

Rounds	0	1	3	5	∞	Δ (0 → ∞)
MT1	28.6	50.6	68.1	79.8	79.8	+51.2
MT2	37.1	71.2	71.4	79.2	79.5	+42.4
MT3	15.1	20.1	40.3	40.8	62.4	+47.3
MT4	15.9	17.4	48.3	50.7	53.3	+37.4
MT5	3.2	3.2	3.2	15.2	29.2	+26.0
MT6	0.5	0.5	1.1	1.9	13.8	+13.3
MT7	0.6	2.3	2.9	2.9	12.6	+12.0
MT8	0.0	0.0	0.0	0.0	0.6	+0.6

4.3.2 Ablation on Re-Planning Rounds

We evaluate our agent on all tasks with increasing maximum rounds of DEPS. The round is defined as a cycle of interactive LLM-based planning with description, explanation, and planning and selecting, i.e., an updated plan. All tasks for every maximum round are executed 30 times and the average success rate is reported in Table 4. We take the vanilla LLM planner as the baseline, in which the model takes the initially generated plan as the final execution plan, without involving any description, re-planning, or self-explanation processes during the task execution. Our results in the previous subsection utilize the maximum rounds possible under maximum tokens capped by OpenAI. We also report the success rate increment from vanilla planner to DEPS of every meta task in column Δ in Table 4. This set of experiments demonstrates that DEPS can iteratively improve its plan in open-world environments. More description, self-explanation, and re-planning rounds produce better results, especially for hard tasks.

4.4 ObtainDiamond Challenge

Mining diamonds in the open-world game Minecraft, i.e. MT8 in Table 2, has been a long-standing challenge for the community [14]. It is challenging because mining diamonds from scratch in Minecraft involves acquiring a sequence of difficult-to-obtain items that require complex planning on goals like mining, inventory management, crafting with and without a crafting table, tool use, smelting iron ingot in a furnace, and mining at the lowest depths. We take the ObtainDiamond task as a bonus experiment to show the capabilities of our zero-shot planner on complex tasks in embodied environments. Previous methods’ success rates on this challenge further vouch for its difficulty. [40, 32] leverages domain-specific reward functions and RL fine-tuning to achieve $\sim 0.1\%$ success rate in 15 minutes of game play. VPT further boosts the success rate to 20% within 20 minutes of play through pre-training on collects $\sim 70k$ hours human demonstrations and finetuning with human-designed reward function [3]. DreamerV3 is trained from scratch to collect diamonds in a modified Minecraft environment (easier to break blocks) with world models to achieve a success rate of 2% [15].

Our DEPS manages to achieve on-par performance in this grand challenge; our agent achieves a 0.59% success rate within 10 minutes of gameplay. Note our method does not specifically fine-tune for this challenge. It is designed to be multi-task in its nature. Furthermore, considering our planner operates with demonstration prompts on a fixed Large Language Model, it can be straightforwardly adapted to other open-ended environments with modifications.

5 Related Works

Task planning with LLMs There have been some methods leveraging the large language model to generate action plans for high-level tasks in embodied environments [42, 9, 11]. [16] decompose natural language commands into sequences of executable actions by text completion and semantic

translation, while SayCan generates feasible plans for robots by jointly decoding an LLM weighted by skill affordances from value functions [4]. For better executing the plan in embodied environments, some methods use an object detector describing the initial environment into the language prompt to produce environment-suitable plans and adopt success detectors to check that each step is executed successfully [17, 20]. [39] and [20] use the pythonic-style prompt to produce more executable plans. However, all of the above methods assume that the initial plan from the LLM is correct. When there are bugs in the initial plan, it’s difficult for the agent to finish the task successfully.

Interactive Planning with LLMs Inner Monologue [17] pilots the front of interactive planning with LLMs, which introduces the feedback (including success detection and scene description) to the planner. However, we found it could still suffer from accumulative planning error, especially in long-horizon open-world tasks. Rather, our “*Describe, Explain, Plan and Select*” (DEPS) method can produce more reliable plans by leveraging chain-of-thought thinking and explanation to locate the errors in previous plans. Moreover, we also propose a goal Selector to further improve the efficiency of the plan, thereby yielding much better performances. Readers are encouraged to refer to the comparative results in Section 4.2 between DEPS and these prior arts. There are also some concurrent works on planning with LLMs [36, 26, 23, 31, 43].

Agents in Minecraft Some previous works have employed the hierarchical architecture to solve long-horizon tasks in Minecraft [29, 27, 24]. Recently, based on the internet-scale corpus, [10] pre-trains a language-conditioned reward function and learns multi-task MineAgent. [3] collects a vast amount of human demonstrations to train a behavior cloning agent. More recently, [15] utilized a learned world model to distill a policy that can efficiently explore in Minecraft. There are also some works focus on learning goal-conditioned policies for better instruction-following [6, 7, 21]. While these efforts all focus on improving the low-level controller. Rather, the planner in our architecture emphasizes applying domain knowledge to propose and arrange the sub-goals. It significantly influences the complexity and breadth of tasks that the agent can handle. Moreover, our planner is zero-shot, making it possible to generalize to other long-horizon open worlds.

6 Limitations

Albeit the impressive results of our approach, we believe there are at least two major limitations within our approach. First of all, our framework relies on privately-held LLMs like GPT-3 and ChatGPT, which makes it less accessible to those who cannot afford or access the service. However, we’re fully committed to ensuring a more democratized method and will explore using open-sourced models including OPT [44] and BLOOM [35]. Another issue is the explicit step-by-step planning in our system. Although it brings us superior performances over the baselines, the planning bottleneck can also prevent our model from being further scaled up. A more appealing approach will be amortizing the planning within an end-to-end trainable goal-conditioned policy, which is worth exploring next. Furthermore, some previous fundamental challenges in planning (e.g., dead ends) may not prevalent in our adopted environments and hence could be inadvertently overlooked by our paper. We are dedicated to addressing more fundamental challenges present in building a multi-task generalist agent in our series of following work.

7 Conclusion

We investigate the problem of planning in open worlds. We identify two major challenges unique to these environments: 1) long-term planning requires precise and multi-step reasoning, and 2) planning efficiency could be compromised since canonical planners do not take the agent’s proximity to parallel goals/subtasks into consideration. We propose “Describe, Explain, Plan and Select” (**DEPS**), an interactive approach based on Large Language Models (LLMs) to tackle them both. Our experiments in the challenging Minecraft domain verify the advantages of our approach over counterparts by marking the milestone of robustly accomplishing 70+ Minecraft tasks and nearly doubling the overall performances. DEPS also is the first planning-based agent that can reach the diamond in this game.

Acknowledgements

This work is funded in part by the National Key R&D Program of China #2022ZD0160301, a grant from CCF-Tencent Rhino-Bird Open Research Fund, NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, an SRA from Meta and a research gift from Amazon Alexa AI, and a gift from RelationalAI. We thank Dai Zhixiang from NVIDIA and Xu Hongming from BIGAI on training LLMs and infrastructure supports, respectively.

References

- [1] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022. 1
- [2] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, 2017. 1
- [3] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*, 2022. 7, 8, 9, 10
- [4] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*, 2022. 1, 2, 3, 5, 10
- [5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 3
- [6] S. Cai, Z. Wang, X. Ma, A. Liu, and Y. Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. *arXiv preprint arXiv:2301.10034*, 2023. 6, 8, 10
- [7] S. Cai, B. Zhang, Z. Wang, X. Ma, A. Liu, and Y. Liang. Groot: Learning to follow instructions by watching gameplay videos. *arXiv preprint arXiv:2310.08235*, 2023. 10
- [8] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 3, 7
- [9] I. Dasgupta, C. Kaeser-Chen, K. Marino, A. Ahuja, S. Babayan, F. Hill, and R. Fergus. Collaborating with language models for embodied reasoning. In *NeurIPS Foundation Models for Decision Making Workshop*, 2022. 9
- [10] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems Datasets and Benchmarks*, 2022. 1, 2, 7, 8, 10
- [11] R. Gong, Q. Huang, X. Ma, H. Vo, Z. Durante, Y. Noda, Z. Zheng, S.-C. Zhu, D. Terzopoulos, L. Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023. 9
- [12] W. H. Guss, M. Y. Castro, S. Devlin, B. Houghton, N. S. Kuno, C. Loomis, S. Milani, S. P. Mohanty, K. Nakata, R. Salakhutdinov, J. Schulman, S. Shiroshita, N. Topin, A. Ummadisingu, and O. Vinyals. The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv: Learning*, 2021. 2
- [13] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al. Neurips 2019 competition: the minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 2019. 2

- [14] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019. 2, 7, 9
- [15] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 9, 10
- [16] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ICML*, 2022. 3, 4, 7, 8, 9
- [17] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022. 2, 3, 5, 7, 8, 10
- [18] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *Ijcai*, pages 4246–4247. Citeseer, 2016. 2, 7
- [19] A. Kanervisto, S. Milani, K. Ramanauskas, N. Topin, Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, W. Yang, W. Hong, Z. Huang, H. Chen, G. Zeng, Y. Lin, V. Micheli, E. Alonso, F. Fleuret, A. Nikulin, Y. Belousov, O. Svidchenko, and A. Shpilman. Minerl diamond 2021 competition: Overview, results, and lessons learned. *neural information processing systems*, 2022. 2
- [20] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. 2, 3, 5, 7, 8, 10
- [21] S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *arXiv preprint arXiv:2306.00937*, 2023. 8, 10
- [22] H. Lin, Z. Wang, J. Ma, and Y. Liang. Mcu: A task-centric framework for open-ended agent evaluation in minecraft. *arXiv preprint arXiv:2310.08367*, 2023. 6, 7, 8
- [23] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023. 10
- [24] Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, and W. Yang. Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:2112.04907*, 2021. 10
- [25] X. Ma, S. Yong, Z. Zheng, Q. Li, Y. Liang, S.-C. Zhu, and S. Huang. Sqa3d: Situated question answering in 3d scenes. *arXiv preprint arXiv:2210.07474*, 2022. 1
- [26] J. Mai, J. Chen, B. Li, G. Qian, M. Elhoseiny, and B. Ghanem. Llm as a robotic brain: Unifying egocentric memory and control. *arXiv preprint arXiv:2304.09349*, 2023. 10
- [27] H. Mao, C. Wang, X. Hao, Y. Mao, Y. Lu, C. Wu, J. Hao, D. Li, and P. Tang. Seihai: A sample-efficient hierarchical ai for the minerl competition. In *Distributed Artificial Intelligence: Third International Conference, DAI 2021, Shanghai, China, December 17–18, 2021, Proceedings* 3, pages 38–51. Springer, 2022. 10
- [28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2
- [29] J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017. 10
- [30] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. 3, 5, 7, 8
- [31] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. 10

- [32] V. P. Patil, M. Hofmarcher, M.-C. Dinu, M. Dorfer, P. M. Blies, J. Brandstetter, J. A. Arjona-Medina, and S. Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution. *ICML*, 2020. 9
- [33] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 5, 6, 8
- [34] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 1
- [35] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022. 10
- [36] N. Shinn, B. Labash, and A. Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023. 10
- [37] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*. PMLR, 2022. 2, 3
- [38] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. 2, 3
- [39] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022. 3, 4, 7, 8, 10
- [40] A. Skrynnik, A. Staroverov, E. Aitygulov, K. Aksenenov, V. Davydov, and A. I. Panov. Forgetful experience replay in hierarchical reinforcement learning from expert demonstrations. *Knowledge-Based Systems*, 218:106844, 2021. 9
- [41] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022. 5, 7, 8
- [42] A. Zeng, A. Wong, S. Welker, K. Choromanski, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022. 2, 9
- [43] C. Zhang, K. Yang, S. Hu, Z. Wang, G. Li, Y. Sun, C. Zhang, Z. Zhang, A. Liu, S.-C. Zhu, et al. Proagent: Building proactive cooperative ai with large language models. *arXiv preprint arXiv:2308.11339*, 2023. 10
- [44] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022. 10