AGENTGEN: Enhancing Planning Abilities for Large Language Model based Agent via Environment and Task Generation

Mengkang Hu[1], **Pu Zhao**[2], **Can Xu**[2], **Qingfeng Sun**[2], **Jianguang Lou**[2], **Qingwei Lin**[2],
Ping Luo[1], **Saravan Rajmohan**[2]
[1]The University of Hong Kong   [2] Microsoft Corporation
{v-humengkang,puzhao}@microsoft.com, pluo.lhi@gmail.com,
{caxu,qins,jlou,qlin,saravar}@microsoft.com

Abstract

Large Language Model (LLM) based agents have garnered significant attention and are becoming increasingly popular. Furthermore, *planning* ability is a crucial component of an LLM-based agent, involving interaction with the *environment* and executing actions to complete a *planning task*, which generally entails achieving a desired goal from an initial state. This paper investigates enhancing the planning abilities of LLM-based agents through instruction tuning, referred to as *agent training*. Recent studies on agent training have demonstrated that utilizing expert-level trajectory data (sequences of action-observation pairs) for instruction-tuning LLMs effectively enhances their planning capabilities. However, existing work primarily focuses on synthesizing trajectories from manually designed planning tasks and environments. The labor-intensive nature of creating these environments and tasks impedes the generation of sufficiently varied and extensive trajectories for agent training. To address this limitation, this paper explores the automated synthesis of diverse environments and a gradual range of planning tasks, from easy to difficult. We introduce a framework, AGENTGEN, that leverages LLMs first to generate environments and subsequently generate planning tasks conditioned on these environments. Specifically, to improve *environmental diversity*, we propose using an inspiration corpus composed of various domain-specific text segments as the context for synthesizing environments. Moreover, to increase the *difficulty diversity* of generated planning tasks, we propose a bidirectional evolution method, BI-EVOL, that evolves planning tasks from easier and harder directions to synthesize a task set with a smoother difficulty curve, thereby enhancing the learning process of LLMs more effectively. These methods collectively contribute to the generation of diverse trajectory data for instruction-tuning. Based on AGENTGEN, we greatly expanded the number of environments and planning tasks available for agent training. The evaluation results from AgentBoard indicate that AGENTGEN greatly enhances the planning capabilities of LLMs. For instance, the AGENTGEN instruction-tuned Llama-3.1-8B outperforms GPT-3.5 in overall performance. Moreover, the AGENTGEN-tuned Llama-3.1-70B model achieves state-of-the-art results in planning tasks. Project page: this URL.

1 Introduction

Recently, owing to advancements in Large Language Models (LLMs) [42, 43, 39, 62], the LLM-based Agents have garnered widespread attention from the artificial intelligence community. Generally, an LLM-based agent refers to utilizing LLMs to perceive the environment, make decisions, and execute actions to substitute or help people accomplish some specific tasks [77, 65, 79]. Furthermore, *planning* is often regarded as one of the most important applications of LLM-based agents, such as

robotic planning [54, 46, 19, 63], travel planning [90, 78], etc. In this study, planning is conceptualized as the systematic process of identifying a sequence of executable actions within a given *environment* to complete a *planning task*, defined as the transition from an initial state to achieve specified goal conditions, considering constraints and available resources [25, 50].

Improving planning capabilities through instruction-tuning LLMs is a significant research problem, referred as *agent training*. As shown in Figure 1, similar to imitation learning [23], a typical agent training process can be divided into three stages: *(i)* Preparing environments and planning tasks. *(ii)* Synthesizing expert-level trajectories (sequences of action-observation pairs) on these planning tasks. For example, utilizing state-of-the-art LLMs (e.g., GPT-4 [43]) as the agent and filtering trajectory based on reward score [86, 6]. *(iii)* Instruction-tuning LLMs with the synthesized trajectory data. Recently, the effectiveness of enhancing the planning capabilities of LLMs through agent training has been demonstrated by many studies [86, 85, 6, 68, 8, 87, 64, 57]. Despite their success, one key limitation of these works is that they primarily rely on manually designed environments and planning tasks. The labor-intensive nature of creating environments and planning tasks hinders the generation of diverse and extensive trajectory data. More explicitly, designing diverse environments requires defining a range of rich and practical scenarios, and implementing these environments typically involves the participation of human experts with programming skills. Additionally, formulating tasks often demands creating a task set with a gradual difficulty progression. Due to this constraint, existing agent training studies typically use only a few environments for data synthesis.
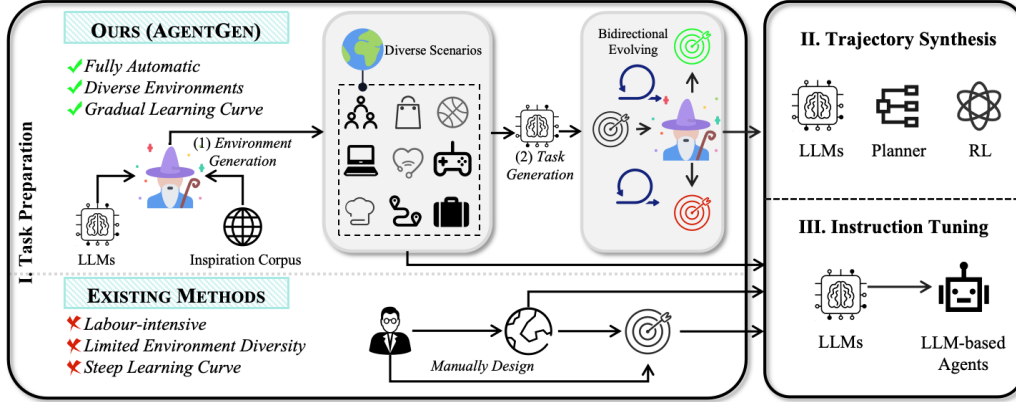

Figure 1: A typical agent training process includes three stages: task preparation, trajectory synthesis, and instruction tuning. AGENTGEN primarily distinguishes itself from existing agent training litera-ture in the task preparation stage, where we introduce a *fully automated* task generation framework AGENTGEN for constructing *diverse* environments and planning tasks with *gradual learning curves*.

To address the aforementioned deficiencies, this paper introduces an automatic framework **AGENTGEN** that utilizes LLMs to construct diverse environments and planning tasks for agent training, expanding the available environments from a few to hundreds. More specifically, AGENTGEN is structured around two stages: (1) **Environment Generation**: Achieving sufficient *environmental diversity* is essential for creating diverse planning tasks, which involves covering a broad range of scenarios and domains. To ensure this, we use an *inspiration corpus* composed of diverse text segments as context for generating environment specifications with LLMs, where actions, restrictions, and other details are defined using natural language. For example, in Figure 2, we randomly selected a text segment from the inspiration corpus: *"How to boost your diet with peanut butter powder?"* This prompted the generation of a related environment specification: *"You are a nutritionist tasked with creating a new healthy recipe book that incorporates peanut butter powder as a key ingredient"*. Subsequently, we prompt the LLM to produce the corresponding code based on this specification, which may be composed of Python, Planning Domain Definition Language (PDDL) [38], or other domain-specific languages. Furthermore, we constructed an environment library to serve as in-context examples and iteratively expanded it by incorporating high-quality newly generated environments. (2) **Task Generation**: Conditioned on the generated environment, we aim to create multiple planning tasks. In this stage, it is crucial to have a gradual set of tasks ranging from easy to difficult, i.e., *difficulty diversity*. To achieve greater difficulty diversity, we propose a bidirectional evolution

method, **BI-EVOL**, where the LLM first generates random planning tasks and then evolves these tasks by applying constraints towards both simplification and increased difficulty. The created task set with BI-EVOL has a smooth difficulty curve that facilitates LLMs' smoother acquisition of planning skills.

To verify the effectiveness of our method, we synthesized environments and planning tasks based on PDDL [38] and constructed a dataset comprising 592 environments, each with 20 tasks. We then used a domain-independent planner to obtain 7,246 high-quality trajectories. Subsequently, we used this trajectory data for instruction-tuning a series of LLMs and demonstrated the trained model on AgentBoard [37]. Since our instruction-tuning dataset is composed of trajectory synthesized from PDDL-based planning tasks, we refer to evaluation tasks implemented in PDDL as *in-domain tasks* and tasks implemented in other programming languages as *out-of-domain tasks*. Importantly, this evaluation was conducted in a *zero-shot* manner without utilizing any trajectory data from these tasks. Experimental results demonstrate that AGENTGEN achieved more than a tenfold improvement over the raw LLama-3.1-8B on in-domain tasks (33.3 vs. 3.0), with overall performance surpassing that of GPT-3.5. Furthermore, the performance of AGENTGEN-tuned Llama-3.1-70B exceeded GPT-4, setting a new state-of-the-art in planning tasks. In out-of-domain tasks, AGENTGEN also demonstrated similar experimental outcomes. Specifically, it led to a significant improvement in average success rates, with the raw LLama-3.1-8B model achieving a 10.0% increase and the 70B model a 3.7% improvement. In summary, the proposed environment and planning task generation method AGENTGEN can help improve planning ability. Moreover, not only can in-domain tasks benefit from this, but out-of-domain tasks also improve, which confirms both the effectiveness and generalization. Our contributions can be summarized as follows:

- We introduce AGENTGEN, which, as far as we know, is the first framework for automatically generating diverse planning tasks and environments targeted for LLM-based agent training.
- We propose utilizing an inspiration corpus as the context for generating environments with LLMs, resulting in 592 diverse environments that encompass a broad range of scenarios.
- We propose a bidirectional evolution method BI-EVOL that evolves seed planning tasks in both simpler and more complex directions, thereby constructing a task set with a smoother difficulty curve.
- We constructed an agent instruction-tuning dataset with 7246 high-quality trajectories through AGENTGEN. LLMs instruction-tuned with this dataset achieved massive improvement in both in-domain and out-of-domain planning tasks, which validated the effectiveness and generalization of AGENTGEN.

2 Preliminary

2.1 Planning Problems

We consider goal-directed deterministic planning problems [50], which are formally defined as a tuple $\mathcal{P} = (\mathbb{T}, \mathbb{E})$, where $\mathbb{E}$ denotes the environment in which the agent interacts and $\mathbb{T}$ denotes the task that the agent needs to complete. Specifically, an environment $\mathbb{E}$ typically models a world, encompassing the definitions of the action space $\mathcal{A}$ and state space $\mathcal{S}$, as well as the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. Task $\mathbb{T}$ is further defined by the tuple $\mathbb{T} = (\mathcal{G}, \mathcal{I})$, where $\mathcal{G}$ refers to the goal conditions and $\mathcal{I}$ refers to initial states of the agent. The initial states $\mathcal{I}$ are a subset of the state space $\mathcal{S}_i$ that specifies the starting conditions of the agent. The goal $\mathcal{G}$ is a subset of the state space $\mathcal{S}_g$ that specifies the desired outcomes or conditions. Specifically, $\mathcal{G}$ can be expressed as $\mathcal{G} = \{s \in \mathcal{S}_g \mid \phi(s) = \text{true}\}$. Here, $\phi(s)$ is a boolean-valued function representing conditions or propositions that must be satisfied for the state s to be considered part of the goal set.

2.2 Planning Problem Implementation

A planning problem can be implemented with programming languages such as Python or domain-specific languages such as Planning Domain Definition Language (PDDL) [38]. For example, in a PDDL-based planning problem, the domain PDDL file can be regarded as the environment $\mathbb{E}$, defining states (predicates) and actions and specifying the transition function using preconditions and effects of each action. The problem PDDL file, on the other hand, can be seen as the task $\mathbb{T}$. Both initial states and goal conditions are typically defined as combinations of predicates. Another widely

used programming language for constructing planning problems is Python. For example, in OpenAI gym[1], a planning problem will be implemented as a Python class, where the transition function is implemented as a method of the class, usually named the "step" or "update" function. Meanwhile, the goal $\mathcal{G}$ is typically represented as a reward function that indicates the objective of the task, and the initial states $\mathcal{I}$ are defined in a method named "reset."

2.3 Large Language Model based Agent

An LLM-based agent leverages a pre-trained language model to operate within the defined environment $\mathbb{E}$ and complete the given task $\mathbb{T}$. Given an environment $\mathbb{E}$, the LLM-based agent perceives its state $\mathcal{S}$ and takes actions $\mathcal{A}$ based on its understanding and processing of the input. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ remains consistent, where the LLM-based agent determines the next state by generating appropriate actions through natural language processing. The goal G guides the LLM-based agent in selecting actions that maximize the reward. The agent utilizes the language model to interpret the task requirements and generate actions that align with achieving the specified goal. In essence, the LLM-based agent forms a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ using the LLM, where $\pi(s)$ is the action taken in state s based on the LLM's understanding and processing of the task.

3 Methodology

Problem Definition   The process of generating planning tasks can be formalized as a function $f : I \rightarrow (\mathbb{T}, \mathbb{E})$, where I is the input space (e.g., instructions or prompts) and tuple $(\mathbb{T}, \mathbb{E})$ is the space of all possible planning tasks and environments. Based on the definition in Section 2.1, we can express this as $f(i) = (\mathbb{T}_i, \mathbb{E}_i), \quad i \in I$, where $\mathbb{T}_i$ is the generated planning task and $\mathbb{E}_i$ is the generated environment for a given input i. Our two-stage approach can be further decomposed as follows: i) **Environment Generation** (§3.1): In the first stage, we generate the environment $\mathbb{E}_i$ based on the input instruction i. This can be represented as $\mathbb{E}_i = g_\mathbb{E}(i)$, where $g_\mathbb{E}$ is the environment generation function that takes the instruction i as input and produces the environment $\mathbb{E}_i$. ii) **Task Generation** (§3.2): In the second stage, we generate the task $\mathbb{T}_i$, conditioned on the environment $\mathbb{E}_i$ generated in the first stage. This can be expressed as: $\mathbb{T}_i = g_\mathbb{T}(i, \mathbb{E}_i)$, where $g_\mathbb{T}$ is the task generation function that takes both the original instruction i and the generated environment $\mathbb{E}_i$ as inputs to produce the task $\mathbb{T}_i$. We will detail the implementation of these two stages in the following section.
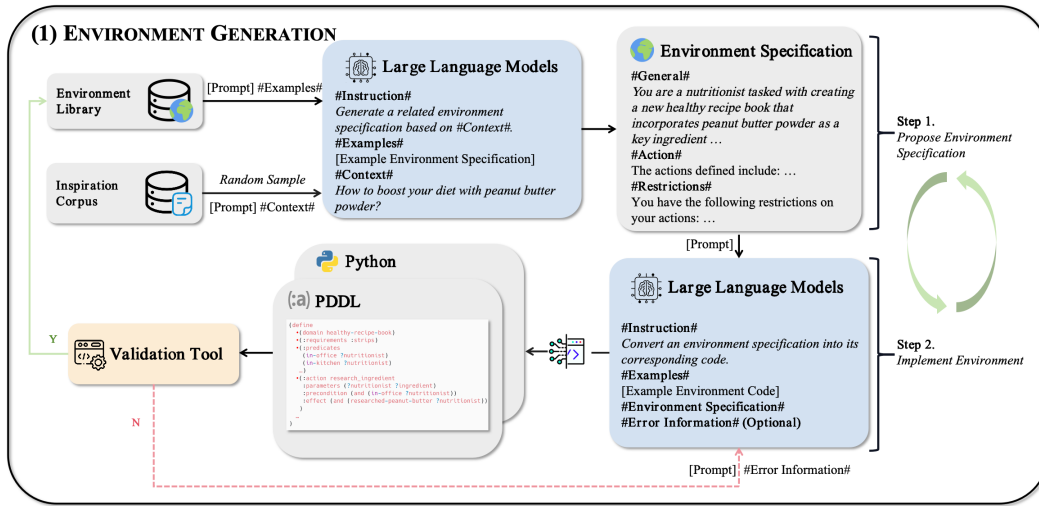
3.1 Environment Generation

Figure 2: Overview of the process of environment generation.

Overview   As is shown in Figure 2, we propose a sophisticated framework for environment generation structured around three main components: (1) an *environment specification generation* module where an LLM first generates a specification of the environment, typically including a general overview of the environment, descriptions of the state space and action space, and definitions of the transition functions; (2) an *environment implementation* module that generates corresponding code based on the environment specification; and (3) an *environment library* that stores previously generated high-quality environments, serving as a comprehensive environment dataset and providing in-context examples for generating new environments. Each component will be elaborated on in the following paragraph.

Environment Specification   We initially prompt the LLM to generate an environment specification, which typically includes an overall depiction of the environment, specific actions and their corresponding preconditions and effects, and certain restrictions within the environment. The environment specification will serve as the basis for generating specific environment codes. This two-stage approach, similar to the Chain-of-Thought [74], can better assist the LLM in creating high-quality environments. For generating environment specifications, One direct approach is to prompt LLMs to generate random environments. However, due to the inherent inductive bias of LLMs, they struggle to generate diverse environments in this way. Therefore, to address this issue, we build an inspiration corpus $D = \{t_0, t_1, \cdot, t_n\}$, containing sufficiently diverse text segments used to serve as the "inspiration" for generating environment specification with LLMs. More specifically, when generating an environment, we first sample a text segment t_i from D, then prompt the LLM to generate a related environment based on t_i. Taking the example in Figure 2, we first sample a text segment "*How to boost your diet with peanut butter powder?*" from D. Then we prompt an LLM to generate a related environment where the agent is defined as a nutritionist tasked with creating a new healthy recipe book that prominently features peanut butter powder as a key ingredient. This approach significantly enhances the diversity of generated environments, thereby empowering more generalized agent training. The inspiration corpus can be implemented in various ways, such as using a large-scale pre-trained corpus like Common Crawl. Alternatively, a domain-specific corpus, such as a code generation dataset [27, 7], can be used to generate environments for a specific domain. This paper uses LIMA [93] as the inspiration corpus, an instruction-tuning dataset with sufficient diversity.

Environment Implementation   Conditioned on the generated environment specification, we generate its corresponding code, i.e., implementing the environment. This can be formulated as a typical code-generation problem with LLMs. We also introduce a validation tool capable of capturing syntax errors to provide feedback during the code generation process, thereby iteratively refining it.

Environment Library   We define the library at iteration t as: $L_t = L_0 \cup \bigcup_{k=1}^{t} \{\mathbb{E}_i | \mathbb{E}_i = g_{\mathbb{E}}(i, L_{k-1}), i \in I_k, v(\mathbb{E}_i) = true\}$, where L_0 is the initial seed library, and the union represents all verified environments generated up to iteration t. This iterative process allows continuous expansion and refinement of the environment library, potentially leading to increasingly complex and diverse environments over time.

3.2   Task Generation

Overview   As depicted in Figure 3, conditioned on the generated environments, we prompt LLMs to generate corresponding planning tasks. We employ a two-stage generation approach BI-EVOL for creating a diverse range of planning tasks in terms of difficulty. We begin by prompting the LLM with a specific environment, enabling it to generate an initial set of planning tasks in a zero-shot way. Subsequently, we adjust these tasks to make them simpler or more challenging, forming a comprehensive set of planning tasks.

Bidirectional Evolution   Many studies have proposed evolving instructions, primarily focusing on making instructions more difficult [80, 36, 35]. The effectiveness of this approach relies heavily on the assumption that LLMs inherently possess the ability to follow simple instructions. However, according to findings from some studies [37, 33], LLMs often exhibit poor performance even in simple planning tasks. Therefore, we propose **BI-EVOL**, which introduces evolution in two directions: easy-evol and hard-evol. Easy-evol typically involves simplifying the goal conditions. The motivation is that easier tasks can facilitate learning when the agent performs poorly and cannot directly learn from typically difficult goals. Conversely, hard-evol usually involves making the goal conditions
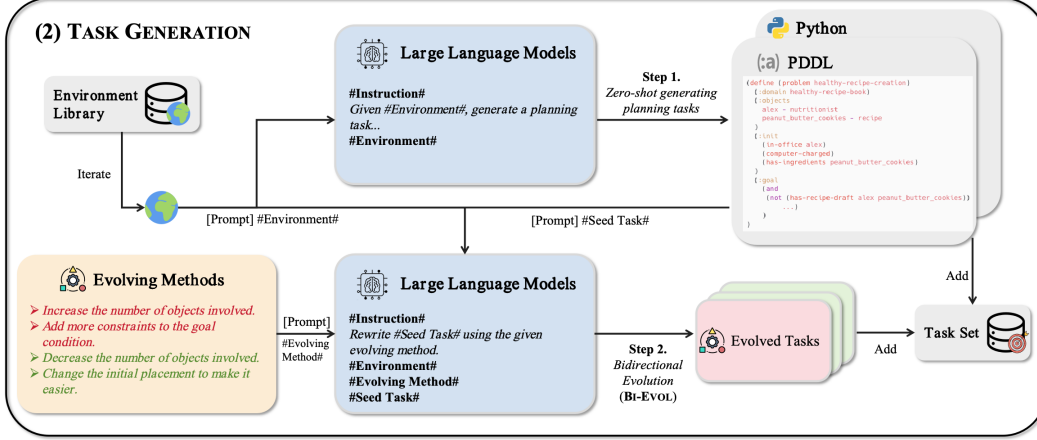
Figure 3: Overview of the process of task generation. The two-stage task generation process includes first generating unconditioned tasks, then applying BI-EVOL to evolve these planning tasks. Ultimately, both parts are incorporated into the task set. In examples of evolving methods, red indicates evolution towards more difficult tasks, while green indicates the opposite.

more complex, increasing the number of steps required for the agent to complete the task. This can further enhance the agent's capability to perform the planning task. To our knowledge, we are the first to introduce bidirectional evolution in the agent data generation scenario. The prompt examples are shown in Figure 3.

4 Experiments

To evaluate the effectiveness of the proposed framework, we synthesize environments and planning tasks using the Planning Domain Definition Language (PDDL), a widely adopted programming language for planning. Subsequently, we evaluate its performance across various unseen planning tasks in a **zero-shot** manner. To validate the effectiveness and generalizability of AGENTGEN, we categorized the evaluated tasks into two distinct groups: i) *In-Domain Tasks*: Planning tasks implemented using PDDL. ii) *Out-of-Domain Tasks*: These comprise tasks developed using other programming languages, such as Python.

4.1 Experimental Setup

Evaluation Tasks For *In-Domain Tasks*, we select four widely used PDDL-based planning tasks: Blocksworld, Gripper, Tyreworld, and Barman [37]. More explicitly, Blocksworld requires an agent to achieve a target configuration by moving blocks, while Gripper involves moving objects between different rooms. Tyreworld simulates changing a car tire, including removing the flat tire, replacing it with a spare, and installing the new tire. Barman emulates a bartender's tasks in mixing cocktails, which include combining various ingredients, using shakers, and garnishing drinks. For *Out-of-Domain Tasks*, we select three challenging partial-observable planning tasks: Alfworld [54] and BabyAI [10], Jericho [15]. Alfworld is an environment designed to test agents' abilities to perform everyday household tasks. While in BabyAI, the agent interprets and executes natural language instructions in a grid-world setting. Jericho [15] is a collection of text-based interactive fiction games in which players issue textual commands to alter the environment.

Evaluation Metrics We utilized two evaluation metrics to evaluate planning ability: *success rate* and *progress rate* [37]. During each interaction round, we assigned a progress rate, denoted as r_t, to measure the progression towards the goal state g. As the agent transitions through states $s_t = [s_0, \ldots, s_t]$, its progress is assessed using a matching score $f(\cdot, g) \to [0, 1]$, which quantifies the similarity between the current state and the goal state. Initially, r_t is set to 0, indicating no progress. Only when the progress rate reaches 1 does the success rate attain 1; all other scenarios yield a 0 outcome. The success rate reflects the agent's capacity to complete a comprehensive task.

Baselines We compare AGENTGEN with a series of widely-used multipurpose foundation models that exhibit state-of-the-art performance, such as GPT-3.5 [42] and GPT-4 [44], CodeLlama [48], Mistral [24], Llama-2 [62], and Llama-3.1 [39]. We use their instruct-tuned versions for all multipurpose foundation models (§A.1). Additionally, some models have undergone specialized training on agent trajectory data, such as AgentLM [86], FireAct [6], Agent-Flan [8]. We also utilize the AgentInstruct [86] dataset to train Llama-3.1, following the training configuration of AGENTGEN as a baseline model.

Implementation Details We followed the environment and task implementation of Agent-Board [37]. For the configuration of evaluation tasks, we employ act-only prompting [82], setting the maximum step length for the LLM agent to 30. We selected LIMA [93] as the text corpus D for generating environments, which leverages various data manipulation techniques to ensure a diverse range of instructions. For environment generation and task generation, we employ GPT-4 [2], configuring the inference parameters with a temperature of 0 and a top_p value of 0.95. Based on AGENTGEN, we generated a total of 592 environments. For each environment, we generated ten unconditioned tasks, which were then evolved into ten refined tasks using BI-EVOL. To generate trajectory data for training, we utilized the domain-independent planner FastDownward[3], ensuring optimal trajectory data. This process ultimately led to 7246 trajectories. More details of the dataset can be found in Appendix B and C. Since the trajectory data is structured, such as *"pickup(o1)"*, we employ GPT-4 to generate a natural language mapping, for example, *"pick up object {arg1}"*, to transform structured actions into natural language actions. We detailed the generation of natural language mapping in A.2. During the training process, we employed Llama-3.1-8B (base version) as our foundation model, blending general instruction data from the ShareGPT dataset in a 1:4 ratio. For the 70B model, we selected Llama-3.1-70B-Instruct and trained it using LoRA [17] without incorporating general instructions, with a rank of 16. The hyperparameters were configured as follows: a batch size of 64, 10 epochs, a context length of 4096 tokens, and no warmup steps. Checkpoints from epochs 5 through 10 were retained and subsequently evaluated on in-domain tasks. The model demonstrating optimal performance was then selected for further evaluation on out-of-domain tasks. We conducted all experiments utilizing V100 and A100 GPUs.

4.2 Evaluation on In-Domain Tasks

As shown in Table 1, the AGENTGEN-tuned Llama-3.1-8B model outperforms GPT-3.5 in overall progress rate (33.3 vs. 25.0). Furthermore, the AGENTGEN-tuned Llama-3.1-70B model slightly surpasses GPT-4 (81.5 vs. 81.2). When compared to other models with similar parameter scales, AGENTGEN consistently demonstrates superior performance across four distinct tasks. In relation to the base Llama-3.1 model, our model exhibits a substantial improvement for both the 8B and 70B versions, with overall progress rates increasing by 30.3 and 2.5, respectively. Notably, in tasks where the success rate of Llama-3.1-8B is zero, AGENTGEN achieves significant breakthroughs, further validating the efficacy of AGENTGEN. From the above, we can draw the following conclusions: *i)* AGENTGEN-tuned Llama-3.1-8B outperforms GPT-3.5 in overall performance, while the 70B version achieves state-of-the-art results; *ii)* AGENTGEN-tuned Llama-3.1 has significantly improved both success rate and progress rate; *iii)* AGENTGEN consistently outperforms other models with similar parameter scales.

4.3 Robustness

To validate the robustness of the constructed dataset with AGENTGEN, we conducted a series of experiments to evaluate its performance across different foundation models. We selected several widely used 7-8B foundation models, including Llama-3-8B, CodeLLama-7B, and Mistral-7B, to test the versatility and effectiveness of AGENTGEN. As is shown in Table 2, all three models exhibited significant improvements after training, with Llama-3-8B showing the highest success rate increase of 10.0 and CodeLlama-7B demonstrating a maximum progress rate increase of 9.9. These experimental results prove that the dataset constructed with AGENTGEN for agent training is highly effective across different models.

---

[2]We applied the gpt-4-20230321 API from Azure OpenAI service.
[3]`https://www.fast-downward.org/`

Table 1: Performance comparison between AGENTGEN and baseline models in in-domain tasks. *"Overall"* is the weighted average of performance in different tasks. "SR" and "PR" stand for "success rate" and "progress rate" metrics.

Model	Size	Version	Gripper		Blockworld		Barman		Tyreworld		Overall	
			SR	PR	SR	PR	SR	PR	SR	PR	SR	PR
GPT-4	–	`2023-05-15`	55.0	83.3	50.0	75.0	75.0	82.5	60.0	80.3	61.7	81.2
	–	`turbo`	50.0	87.8	40.0	71.7	10.0	17.5	10.0	39.3	23.3	44.7
GPT-3.5	–	`turbo`	0.0	30.6	0.0	18.3	10.0	21.7	10.0	27.1	5.0	25.0
	–	`turbo-16k`	0.0	28.2	0.0	20.0	5.0	13.3	10.0	32.7	3.3	22.6
CODELLAMA	7B	`instruct`	0.0	7.4	0.0	8.3	0.0	0.0	10.0	26.0	1.7	8.2
	13B	`instruct`	5.0	15.6	0.0	5.0	0.0	0.0	0.0	19.3	1.7	9.3
	34B	`instruct`	0.0	28.7	10.0	21.7	5.0	7.5	0.0	17.1	3.3	18.5
MISTRAL	7B	`instruct-v0.2`	0.0	5.3	0.0	10.0	0.0	2.5	0.0	7.3	0.0	5.5
LLAMA-2	7B	`chat`	0.0	1.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5
	13B	`chat`	0.0	0.0	0.0	6.7	0.0	1.7	0.0	14.8	0.0	4.1
	70B	`chat`	0.0	8.8	0.0	5.0	5.0	9.2	0.0	7.8	1.7	8.1
FIREACT	7B	–	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	1.5
AGENT-FLAN	7B	–	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AGENTLM	7B	–	0.0	0.0	0.0	0.0	0.0	2.5	0.0	0.0	0.0	0.8
	70B	–	0.0	0.8	0.0	6.7	5.0	13.3	10.0	26.0	3.3	10.2
LLAMA-3.1	8B	`instruct`	0.0	0.0	0.0	1.7	0.0	0.0	0.0	16.2	0	3.0
w. AGENTINSTRUCT	8B	–	0.0	3.4	0.0	5.0	0.0	6.7	10.0	26.0	1.7	8.5
w. AGENTGEN	8B	–	20.0	45.2	20.0	31.7	10.0	32.7	10.0	32.7	15.0	33.3
LLAMA-3.1	70B	`instruct`	55.0	89.3	50.0	70.0	70.0	80.0	40.0	65.3	56.7	79.0
w. AGENTINSTRUCT	70B	–	45.0	78.9	70.0	80.0	25.0	32.5	50.0	74.4	43.4	62.9
w. AGENTGEN	70B	–	55.0	89.3	50.0	63.3	70.0	82.5	50.0	82.2	58.3	81.5

Table 2: Overall performance comparison of models before and after training with AGENTGEN on in-domain tasks. "SR" and "PR" stands for "success rate" and "progress rate" respectively.

Model	Before		After		Δ	
	SR	PR	SR	PR	SR	PR
Llama-3-8B	1.7	13.4	11.7	23.0	10.0	9.6
CodeLlama-7B	1.7	8.2	6.7	18.1	5.0	9.9
Mistral-7B	0.0	5.5	1.7	10.4	1.7	4.9

4.4 Evaluation on Out-of-Domain Tasks

We also conducted evaluations on out-of-domain agent tasks. As illustrated in Table 3, similar experimental phenomena were observed. Firstly, AGENTGEN demonstrates a substantial performance improvement over Llama-3.1, with an increase of 13.1% in the average progress rate for the 8B model and 5.0% for the 70B model. Additionally, the AGENTGEN-tuned Llama-3.1-8B model outperforms GPT-3.5. When compared to general models and agent fine-tuning models with similar parameter scales, AGENTGEN consistently outperforms them on both tasks. The superior performance on out-of-domain tasks further emphasizes the effectiveness and generalization capability of our data synthesis methods.

5 Related Work

Large Language Model based Agent. Large Language Models have demonstrated exceptional reasoning capabilities [62, 39, 42, 43, 24]. Owing to such abilities, over the past two years, LLM-based agents have experienced significant development [53, 75, 16, 58, 65, 77]. Unlike the traditional method of using LLMs for text-based reasoning, such as Chain-of-Thought [74], LLM-based agents typically involve interaction with the environment, adjusting the output in a closed-loop manner based on environmental information. These LLM-based agents, now fortified with capabilities like Memorizing [91, 32, 29, 84, 53, 88, 95, 61, 22], Tool-use [9, 45, 52, 28, 51, 47], and Planning [12, 5, 41, 40, 49, 2], exhibit a marked enhancement in their overall efficacy. Although this paper mainly

Table 3: Performance comparison between AGENTGEN and baseline models on out-of-domain tasks. "SR" and "PR" stand for "success rate" and "progress rate" metrics.

Model	Size	Version	Alfworld [54]		BabyAI [10]		Jericho [15]		Average	
			SR	PR	SR	PR	SR	PR	SR	PR
GPT-4	–	2023-05-15	43.4	65.5	56.2	70.7	35.0	52.4	44.9	62.9
GPT-3.5	–	turbo	17.2	35.6	18.9	31.9	0.0	20.4	12.0	29.3
	–	turbo-16k	4.5	25.2	33.9	45.1	0.0	16.1	12.8	28.8
CODELLAMA	7B	instruct	1.4	2.2	15.2	28.3	0.0	9.2	5.5	13.9
	13B	instruct	2.2	13.4	17.0	22.2	0.0	0.0	6.4	11.9
	34B	instruct	3.0	11.3	13.4	19.9	0.0	15.5	5.5	15.6
MISTRAL	7B	instruct-v0.2	0.0	9.8	18.1	24.4	0.0	12.1	6.0	15.4
LLAMA-2	7B	chat	0.0	1.5	5.4	8.3	0.0	4.2	1.8	4.7
	13B	chat	0.0	7.8	6.2	18.2	0.0	3.2	2.1	9.7
	70B	chat	3.0	13.2	19.6	30.0	0.0	7.8	7.5	17.0
FIREACT	7B	–	0.0	0.8	4.5	8.6	0.0	2.8	1.5	4.7
AGENT-FLAN	7B	–	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.3
AGENTLM[†]	7B	–	–	–	8.0	9.9	5.5	15.2	–	–
	70B	–	–	–	27.7	37.1	0.0	18.4	–	–
LLAMA-3.1	8B	instruct	0.0	10.5	17.9	33.6	0.0	8.8	6.0	17.6
w. AGENTTUNING†	8B	–	–	–	10.7	19.3	0.0	8.2	–	–
w. AGENTGEN	8B	–	17.9	31.7	32.1	46.0	0.0	14.3	16.0	30.7
LLAMA-3.1	70B	instruct	17.2	42.7	38.4	57.2	10.0	31.5	21.8	43.8
w. AGENTTUNING†	70B	–	–	–	17.9	35.9	10.0	31.9	–	–
w. AGENTGEN	70B	–	19.4	46.1	42.0	62.2	15.0	38.1	25.5	48.8

focuses on the planning capability of LLM-based agents, we believe AGENTGEN has the potential to generalize to other scenarios of LLM-based agents.

Planning with Large Language Models. Planning is one of the key applications of LLM-based agents, applicable in various scenarios such as robotic planning [54, 46, 19, 63, 11, 76, 31, 13], travel planning [78, 1], calendar scheduling [90], code generation [4] and others [72]. It is typically defined as the process of systematically determining a sequence of actions or steps required to achieve a desired goal from an initial state, considering constraints and available resources. This definition primarily differentiates from studies that utilize LLMs to generate ungrounded plans as guidance for problem-solving [94, 66], rather than directly producing executable actions. Planning can be categorized into two types: open-loop planning, where the LLM outputs an entire action sequence before execution [19, 63], and closed-loop planning, where the LLM-based agent decides the next action based on real-time environmental interaction after executing a previous action [55, 5, 59, 60, 30, 56, 21, 20]. This paper mainly focuses on close-loop planning, which is more adaptable for error correction, human interaction, and environmental grounding. Recent studies on close-loop planning have integrated chain-of-thought reasoning into the planning process [82]. Additionally, some papers have explored the use of tree-search methods to enhance the performance of LLM planning [18, 14, 83, 34, 89, 70, 92]. Instead of designing novel frameworks or engaging in prompt engineering, this paper explores how training can enhance the planning capabilities of LLM-based agents.

Agent Training. Recently, numerous studies have aimed to enhance LLM-based agent capabilities by incorporating agent trajectory data into their training [68, 8, 87, 64, 57]. Advanced works such as AgentTuning [86] utilize GPT-4 to generate trajectory data across six distinct environments. Subsequently, this data is filtered and employed in training Large Language Models, enhancing the agent capabilities of base models. Another work, FireAct [6], proposes training with both CoT data

---

[3]†AgentTuning utilized Alfworld's training set, meaning Alfworld cannot be considered an out-of-domain task. Consequently, we did not evaluate the performance of AgentLM or the AgentTuning-trained model on Alfworld.

and ReAct format data, enabling the model to discern when to use reasoning to solve problems and when to call external tools. Agent LUMOS [85] suggests separately training Planning and Grounding models, enabling LLM-based agents to learn to decompose complex problems before execution. LLM-Modulo framwork [26] proposes to leverage LLMs generating candidate plans and verify them with an external verifier. Then, use the verified trajectories for fine-tuning LLMs. Similarly, [3] takes a generate-test loop to synthesize trajectories for LLM training. Unlike previous papers on all agent training, AGENTGEN goes beyond merely generating trajectory data using Large Language Models. Instead, we utilize Large Language Models to generate agent environments, which can be considered a more foundational application. As a result, we have constructed over 500 environments for training, whereas previous works typically use fewer than 10 environments to synthesize agent data.

Environment and Task Generation with Large Language Models. The utilization of LLMs to generate environments and tasks is an emerging application. Some studies have explored utilizing LLMs to generate layouts in robotic simulations, typically involving the creation of configuration files [71, 81, 67]. While these methods can construct numerous scene-level environments, they often struggle to achieve diversity at the underlying mechanism level. AgentTuning [86] employs a task generation approach similar to the Self-instruct [73] method, using the test set as seed data. This approach not only poses a risk of data leakage but also leads to insufficient diversity in task difficulty. ByteSized32 [69] uses LLMs to generate Python-based games based on predefined task specifications automatically. Similarly, other works [13] leverage LLMs to automatically construct PDDL domains based on a task specification. In contrast to these studies, this paper proposes using a diverse text corpus to generate environment code automatically. This approach facilitates the creation of a wide range of rich environments without predefined definitions.

6 Conclusion

In this paper, we explore using LLMs to automatically generate environment and planning tasks for LLM-based agent training. Specifically, for generating diverse environments, we propose utilizing an inspiration corpus composed of various domain-specific text segments as the context for environment synthesis. To enhance the difficulty diversity of generated planning tasks, we introduce a bidirectional evolution method, BI-EVOL, which evolves planning tasks from both easier and more challenging directions to create a task set with a more gradual difficulty curve, thereby improving the effectiveness of LLM learning. Based on AGENTGEN, we developed a dataset consisting of 592 environments and 7246 trajectories and trained it on a series of LLMs. The AGENTGEN-tuned Llama-3.1-8B model surpassed GPT-3.5 on planning tasks, while the AGENTGEN-tuned Llama-3.1-70B model achieved a new state-of-the-art performance.

References

[1] Mohamed Aghzal, Erion Plaku, and Ziyu Yao. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. *arXiv preprint arXiv:2310.03249*, 2023.

[2] Anurag Ajay, Seungwook Han, Yilun Du, Shuang Li, Abhi Gupta, Tommi Jaakkola, Josh Tenenbaum, Leslie Kaelbling, Akash Srivastava, and Pulkit Agrawal. Compositional foundation models for hierarchical planning. *Advances in Neural Information Processing Systems*, 36, 2024.

[3] Daman Arora and Subbarao Kambhampati. Learning and leveraging verifiers to improve planning capabilities of pre-trained language models. *arXiv preprint arXiv:2305.17077*, 2023.

[4] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B Ashok, and Shashank Shet. Codeplan: Repository-level coding using llms and planning. *Proceedings of the ACM on Software Engineering*, 1(FSE):675–698, 2024.

[5] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.

[6] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*, 2024.

[9] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*, 2022.

[10] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

[11] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. Task and motion planning with large language models for object rearrangement, 2023.

[12] Zeyu Gao, Yao Mu, Jinye Qu, Mengkang Hu, Lingyue Guo, Ping Luo, and Yanfeng Lu. Dagplan: Generating directed acyclic dependency graphs for dual-arm cooperative planning. *arXiv preprint arXiv:2406.09953*, 2024.

[13] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

[14] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

[15] Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910, 2020.

[16] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. MetaGPT: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[17] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[18] Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*, 2023.

[19] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[20] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.

[21] Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, and Brian Ichter. Grounded decoding: Guiding text generation with grounded models for robot control, 2023.

[22] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. Recommender ai agent: Integrating large language models for interactive recommendations. *arXiv preprint arXiv:2308.16505*, 2023.

[23] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[24] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[25] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391.

[26] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Kaya Stechly, Mudit Verma, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*, 2024.

[27] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.

[28] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

[29] Xinnian Liang, Bing Wang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zejun Ma, and Zhoujun Li. Unleashing infinite-length input capacity for large-scale language models with self-controlled memory system. *arXiv e-prints*, pages arXiv–2304, 2023.

[30] Bill Yuchen Lin, Chengsong Huang, Qian Liu, Wenda Gu, Sam Sommerer, and Xiang Ren. On grounded planning for embodied tasks with language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13192–13200, 2023.

[31] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[32] Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023.

[33] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[34] Yanming Liu, Xinyue Peng, Yuwei Zhang, Jiannan Cao, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. Tool-planner: Dynamic solution tree planning for large language model with tool clustering. *arXiv preprint arXiv:2406.03807*, 2024.

[35] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

[36] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

[37] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024.

[38] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998. URL `https://api.semanticscholar.org/CorpusID:59656859`.

[39] Meta AI. Introducing meta Llama 3: The most capable openly available LLM to date, April 2024. URL `https://ai.meta.com/blog/meta-llama-3/`. Accessed: 2024-04-18.

[40] Yao Mu, Junting Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. *arXiv preprint arXiv:2402.16117*, 2024.

[41] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024.

[42] OpenAI. Openai: Introducing chatgpt, 2022. URL `https://openai.com/blog/chatgpt`.

[43] OpenAI. Gpt-4 technical report, 2023.

[44] R OpenAI. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2:13, 2023.

[45] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

[46] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.

[47] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[48] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

[49] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*, 2023.

[50] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[51] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettle-moyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *CoRR*, abs/2302.04761, 2023. doi: 10.48550/ARXIV.2302.04761. URL https://doi.org/10.48550/arXiv.2302.04761.

[52] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hug-ginggpt: Solving AI tasks with chatgpt and its friends in huggingface. *CoRR*, abs/2303.17580, 2023. doi: 10.48550/ARXIV.2303.17580. URL https://doi.org/10.48550/arXiv.2303.17580.

[53] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[54] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[55] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[56] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2023.

[57] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*, 2024.

[58] Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

[59] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023.

[60] Simeng Sun, Yang Liu, Shuohang Wang, Chenguang Zhu, and Mohit Iyyer. Pearl: Prompting large language models to plan and execute actions over long documents. *arXiv preprint arXiv:2305.14564*, 2023.

[61] Jihoon Tack, Jaehyung Kim, Eric Mitchell, Jinwoo Shin, Yee Whye Teh, and Jonathan Richard Schwarz. Online adaptation of language models with a memory of amortized contexts. *arXiv preprint arXiv:2403.04317*, 2024.

[62] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[63] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36, 2024.

[64] Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. Llms in the imag-inarium: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*, 2024.

[65] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.

[66] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

[67] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023.

[68] Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*, 2024.

[69] Ruoyao Wang, Graham Todd, Eric Yuan, Ziang Xiao, Marc-Alexandre Côté, and Peter Jansen. Bytesized32: A corpus and challenge task for generating task-specific world models expressed as text games. *arXiv preprint arXiv:2305.14879*, 2023.

[70] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[71] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. *arXiv preprint arXiv:2311.01455*, 2023.

[72] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, 2023.

[73] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[74] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[75] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[76] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied task planning with large language models, 2023.

[77] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

[78] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*, 2024.

[79] Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, Leo Z. Liu, Yiheng Xu, Hongjin Su, Dongchan Shin, Caiming Xiong, and Tao Yu. Openagents: An open platform for language agents in the wild. *CoRR*, abs/2310.10634, 2023. doi: 10.48550/ARXIV.2310.10634. URL `https://doi.org/10.48550/arXiv.2310.10634`.

[80] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[81] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16227–16237, 2024.

[82] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[83] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[84] Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, et al. Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151*, 2023.

[85] Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*, 2023.

[86] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

[87] Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*, 2024.

[88] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[89] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36, 2024.

[90] Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, et al. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*, 2024.

[91] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

[92] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

[93] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

[94] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[95] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 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*, 2023.

A More Implementation Details

A.1 Models

We applied the instruct version for models. Specifically, the detailed version for each model is presented in Table 4.

Model	Version
CodeLlama	meta-llama/CodeLlama-7b-Instruct-hf
Mistral	mistralai/Mistral-7B-Instruct-v0.2
Llama2	meta-llama/Llama-2-7b-chat-hf
Llama3	meta-llama/Meta-Llama-3-8B-Instruct
AgentLM	THUDM/agentlm-7b

Table 4: Evaluated models in this study.

A.2 Natural Language Mapping

We leverage GPT-4 to generate the natural language mapping that converts structured actions into its natural language format. When the mapping failed to yield, we heuristically serialized the structured actions. The prompt for generating natural language mapping with GPT-4 is as follows:

Natural Language Mapping Generation

I would like you to create natural language mapping for PDDL.
The form of the natural language mapping is a Python dictionary, wherein
1. The key corresponds to the name of a predicate or action within the domain PDDL.
2. The value is its equivalent in natural language, with parameters presented in "{argn}", where n is the index of its parameters in the PDDL expression.
3. You must ensure that the number of "{}" corresponds precisely to the number of parameters in predicates or actions.
4. You should very carefully check the order of {argn}.

Your output must strictly follow the provided example.

Example:
PDDL Domain:
```pddl
(define (domain hanoi)
(:requirements :strips)
(:predicates
(clear ?x)
(on ?x ?y)
(smaller ?x ?y)
)

(:action move
:parameters (?disc ?from ?to)
:precondition (and (smaller ?to ?disc) (on ?disc ?from)
(clear ?disc) (clear ?to))
:effect (and (clear ?from) (on ?disc ?to) (not (on ?disc ?from))
(not (clear ?to))))
)
```
Specification:
Your goal is to solve the Tower of Hanoi puzzle, which involves moving a stack of discs from one peg to another, with the restriction that no disc may be placed on top of a smaller disc.

The puzzle is solved when all the discs are moved to the target peg following these rules.

The actions defined in this domain include:
- move <disc> <from> <to>: This action allows moving a disc from one peg to another. The preconditions for this action are that the target peg is smaller than the disc being moved, the disc is on the source peg, and both the disc and the target peg are clear (i.e., there is no disc on top of them). The effect of this action is that the source peg becomes clear, the disc is now on the target peg, the disc is no longer on the source peg, and the target peg is no longer clear.

You have the following restrictions on your actions:
- A disc can only be moved if it is clear, meaning there is no other disc on top of it.
- A disc can only be placed on another disc or peg that is larger than itself.
- A disc can only be moved to a peg that is clear.
- Once a disc is moved from a peg, that peg becomes clear.
- Once a disc is placed on a peg, that peg is no longer clear.
Natural Language Mapping:
```python
{
"clear": "{arg1} is clear.",
"on": "{arg1} is on {arg2}.",
"smaller": "{arg1} is smaller than {arg2}.",
"move": "Move {arg1} from {arg2} to {arg3}."
}
```

You need to generate the corresponding natural language mapping for the following pddl domain.

PDDL Domain:
{PDDL_Domain}
Specification:
{PDDL_Description}
Natural Language Mapping:

B More Statistics on Environment

B.1 Environment Specification

We analyzed the token distribution within the environmental specifications. Among the 592 environmental specifications, the average token count is 473.55, with a median of 467.00. The minimum token count is 207, and the maximum is 934. As depicted in Figure 4, the number of specification tokens for the environment is predominantly concentrated within the range of 300 to 699.

B.2 Environment Implementation

The scale of action space and state space in an environment typically dictates its complexity, with a greater number of actions and states generally indicating a more complex environment. An environment library with a greater variety of difficulty levels is preferable for a training set. As shown in Figure 5, there is a significant diversity in the number of actions and predicates.

B.3 Diversity Analysis

We evaluate the diversity of generated environments using cosine similarity. More specifically, we randomly sampled 100 environment specifications for better visualization and converted them into TF-IDF vectors. After calculating the cosine similarity matrix between all pairs of specifications, we visualize the matrix using heatmap as is shown in Figure 6. The computed average cosine similarity
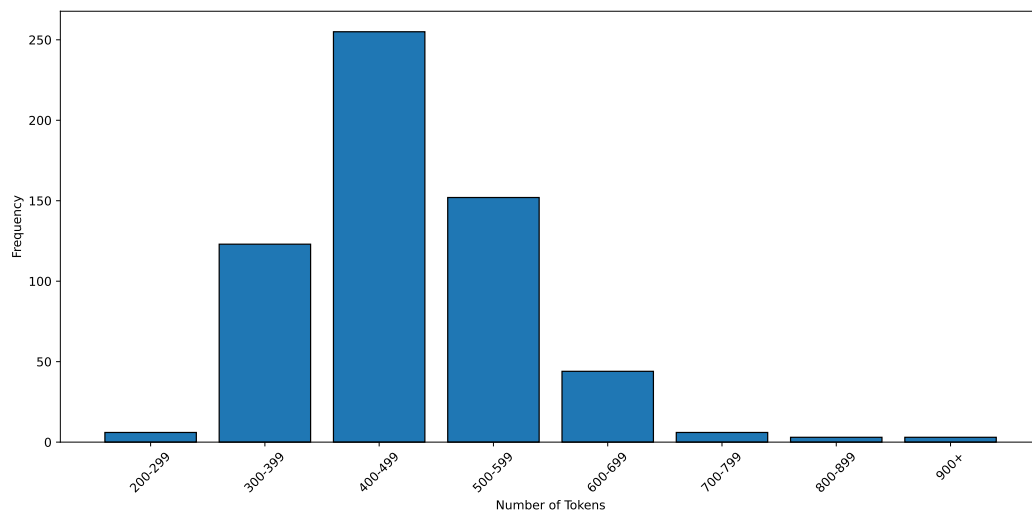
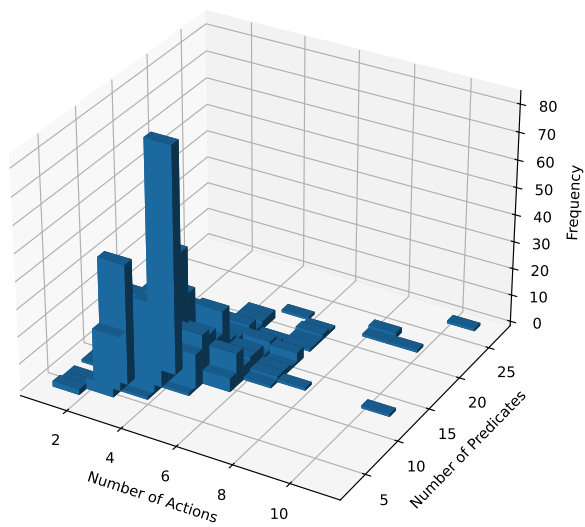Figure 4: The token distribution of the generated environment specification.

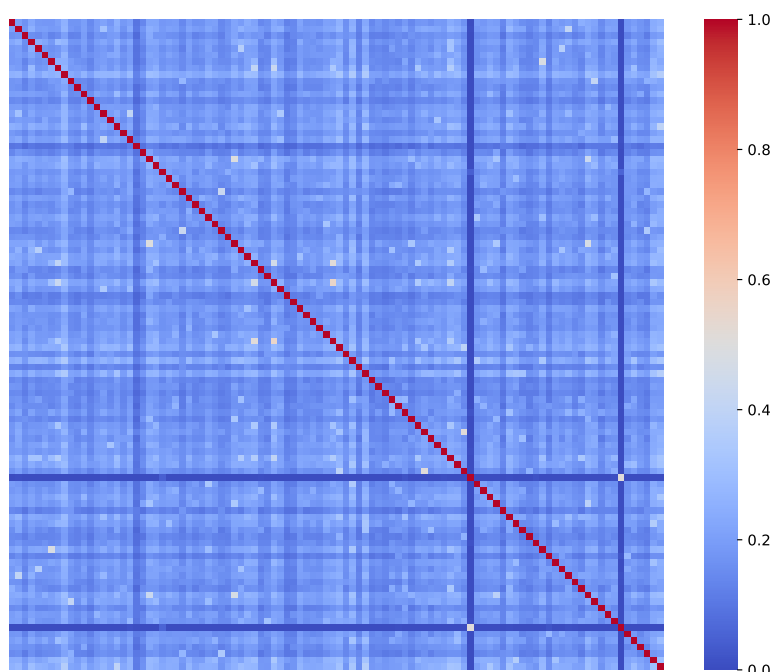Figure 5: The frequency distribution of actions and predicates in datasets.

Figure 6: Cosine similarity heatmap depicting the semantic relationships among randomly sampled 100 environment specifications. Darker shades represent a higher similarity between the two specifications.

of the sampled environment specifications is 0.176, indicating that the corpus exhibits a high degree of diversity, reflecting a rich tapestry of distinct semantic features and thematic elements.

C Examples

In this section, we present the specific details of the cases depicted in Figure 2 and Figure 3.

C.1 Environment Specification

> **Environment Specification**
>
> You are a nutritionist tasked with creating a new healthy recipe book that incorporates peanut butter powder as a key ingredient. Your environment consists of your office, a computer for research and writing, a kitchen for testing recipes, and a collection of ingredients including peanut butter powder. The computer can be used to write and save recipes, but it needs to be charged after every three hours of use.
>
> The actions defined in this domain include:
> - research_ingredient <location> <ingredient> <nutritionist>: This action allows the nutritionist to research peanut butter powder at the office. The effect of this action is that the nutritionist gains knowledge about the nutritional benefits and potential uses of peanut butter powder in recipes.

- develop_recipe <location> <computer> <nutritionist> <ingredient>: This action allows the nutritionist to create a new recipe using peanut butter powder. The precondition for this action is that the nutritionist has researched peanut butter powder. The effect of this action is a draft of a new recipe that incorporates peanut butter powder.
- test_recipe <location> <kitchen> <nutritionist> <recipe>: This action allows the nutritionist to test the drafted recipe in the kitchen. The preconditions for this action are that the nutritionist has developed a recipe and has all the necessary ingredients, including peanut butter powder. The effect of this action is feedback on the recipe's taste, nutritional value, and overall feasibility.
- finalize_recipe <location> <computer> <nutritionist> <tested_recipe>: This action allows the nutritionist to finalize the recipe after testing and receiving feedback. The preconditions for this action are that the nutritionist has tested the recipe and the computer is charged. The effect of this action is a completed recipe that is ready to be included in the healthy recipe book.

You have the following restrictions on your actions:
- You can only research peanut butter powder if you are in the office where your research materials are located.
- You can only develop a recipe after researching peanut butter powder and must have a charged computer.
- You can only test a recipe in the kitchen if you have developed a recipe and have all the necessary ingredients.
- You can only finalize a recipe after testing it in the kitchen and receiving feedback, and if your computer is charged.

C.2   Environment Implementation

```
(define (domain healthy-recipe-book)
  (:requirements :strips)
  (:predicates
    (in-office ?nutritionist)
    (in-kitchen ?nutritionist)
    (researched-peanut-butter ?nutritionist)
    (has-recipe-draft ?nutritionist ?recipe)
    (has-tested-recipe ?nutritionist ?recipe)
    (computer-charged)
    (has-ingredients ?recipe)
  )

  (:action research_ingredient
    :parameters (?nutritionist ?ingredient)
    :precondition (and
                  (in-office ?nutritionist))
    :effect (and
            (researched-peanut-butter ?nutritionist))
  )

  (:action develop_recipe
    :parameters (?nutritionist ?ingredient)
    :precondition (and
                  (researched-peanut-butter ?nutritionist)
                  (computer-charged))
    :effect (and
            (has-recipe-draft ?nutritionist ?ingredient)
            (not (computer-charged)))
  )

  (:action test_recipe
    :parameters (?nutritionist ?recipe)
    :precondition (and
                  (in-kitchen ?nutritionist)
                  (has-recipe-draft ?nutritionist ?recipe)
                  (has-ingredients ?recipe))
    :effect (and
            (has-tested-recipe ?nutritionist ?recipe))
  )

  (:action finalize_recipe
    :parameters (?nutritionist ?tested_recipe)
    :precondition (and
                  (has-tested-recipe ?nutritionist ?tested_recipe)
                  (computer-charged))
    :effect (and
            (not (computer-charged))
            (not (has-tested-recipe ?nutritionist ?tested_recipe)))
  )
```

C.3 Trajectory Data

(User) You are a nutritionist tasked with creating a new healthy recipe book that incorporates peanut butter powder as a key ingredient. Your environment consists of your office, a computer for research and writing, a kitchen for testing recipes, and a collection of ingredients including peanut butter powder. The computer can be used to write and save recipes, but it

needs to be charged after every three hours of use.

The actions defined in this domain include:
- research_ingredient <location> <ingredient> <nutritionist>: This action allows the nutritionist to research peanut butter powder at the office. The effect of this action is that the nutritionist gains knowledge about the nutritional benefits and potential uses of peanut butter powder in recipes.
- develop_recipe <location> <computer> <nutritionist> <ingredient>: This action allows the nutritionist to create a new recipe using peanut butter powder. The precondition for this action is that the nutritionist has researched peanut butter powder. The effect of this action is a draft of a new recipe that incorporates peanut butter powder.
- test_recipe <location> <kitchen> <nutritionist> <recipe>: This action allows the nutritionist to test the drafted recipe in the kitchen. The preconditions for this action are that the nutritionist has developed a recipe and has all the necessary ingredients, including peanut butter powder. The effect of this action is feedback on the recipe's taste, nutritional value, and overall feasibility.
- finalize_recipe <location> <computer> <nutritionist> <tested_recipe>: This action allows the nutritionist to finalize the recipe after testing and receiving feedback. The preconditions for this action are that the nutritionist has tested the recipe and the computer is charged. The effect of this action is a completed recipe that is ready to be included in the healthy recipe book.

You have the following restrictions on your actions:
- You can only research peanut butter powder if you are in the office where your research materials are located.
- You can only develop a recipe after researching peanut butter powder and must have a charged computer.
- You can only test a recipe in the kitchen if you have developed a recipe and have all the necessary ingredients.
- You can only finalize a recipe after testing it in the kitchen and receiving feedback, and if your computer is charged.
Goal: The goal is to satisfy the following conditions: The computer is charged. jordan has tested the recipe almond_butter_bars.
Observation:The computer is charged. The recipe almond_butter_bars has all the necessary ingredients. jordan has a recipe draft for almond_butter_bars. jordan has researched peanut butter. jordan is in the kitchen.
(Assistant) Action: jordan tests the recipe almond_butter_bars.
(User) Observation: The computer is charged. The recipe almond_butter_bars has all the necessary ingredients. jordan has a recipe draft for almond_butter_bars. jordan has researched peanut butter. jordan has tested the recipe almond_butter_bars. jordan is in the kitchen.
(Assistant) Action: jordan develops a recipe using almond_butter_bars.