arXiv:2303.16563v1 [cs.LG] 29 Mar 2023

Plan4MC: Skill Reinforcement Learning and Planning for Open-World Minecraft Tasks

Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie,
Penglin Cai, Hao Dong, Zongqing Lu[†]

PKU BAAI

Abstract

We study building a multi-task agent in Minecraft. Without human demonstrations, solving long-horizon tasks in this open-ended environment with reinforcement learning (RL) is extremely sample inefficient. To tackle the challenge, we decompose solving Minecraft tasks into learning basic skills and planning over the skills. We propose three types of fine-grained basic skills in Minecraft, and use RL with intrinsic rewards to accomplish basic skills with high success rates. For skill planning, we use Large Language Models to find the relationships between skills and build a skill graph in advance. When the agent is solving a task, our skill search algorithm walks on the skill graph and generates the proper skill plans for the agent. In experiments, our method accomplishes 24 diverse Minecraft tasks, where many tasks require sequentially executing for more than 10 skills. Our method outperforms baselines in most tasks by a large margin. The project's website and code can be found at https://sites.google.com/view/plan4mc.

1 Introduction

Learning diverse tasks in open-ended worlds is a significant milestone toward building generally capable agents. Recent studies in multi-task reinforcement learning (RL) have achieved great successes in many narrow domains like games [26] and robotics [33]. However, transferring prior methods to open-ended domains [29, 8] remains unexplored. Minecraft, a popular open-world game with an infinitely large world size and a huge variety of tasks, has been regarded as a challenging benchmark [9, 8].

Previous works usually build policies in Minecraft upon imitation learning, which requires expert demonstrations [9, 4, 31] or large-scale video datasets [2]. Without demonstrations, RL in Minecraft is extremely sample-inefficient. A state-of-the-art model-based method [11] takes over 10M environmental steps to harvest cobblestones , even if the block breaking speed of the game simulator is set to very fast additionally. This difficulty comes from at least two aspects. First, the world size is too large and the requisite resources are distributed far away from the agent. With partially observed visual input, the agent cannot identify its state or do effective exploration easily. Second, a task in Minecraft usually has a long horizon, with many sub-goals. For example, mining a cobblestone involves more than 10 sub-goals (from harvesting logs to crafting wooden pickaxes) and requires thousands of environmental steps.

To mitigate this issue, we propose to solve diverse tasks in a hierarchical fashion. In Minecraft, we define a set of basic skills. Then, solving a task can be decomposed into planning for a proper sequence of basic skills and executing the skills interactively.

[†] Correspondence to Zongqing Lu <zongqing.lu@pku.edu.cn>, Haoqi Yuan <yhq@pku.edu.cn>

Preprint.

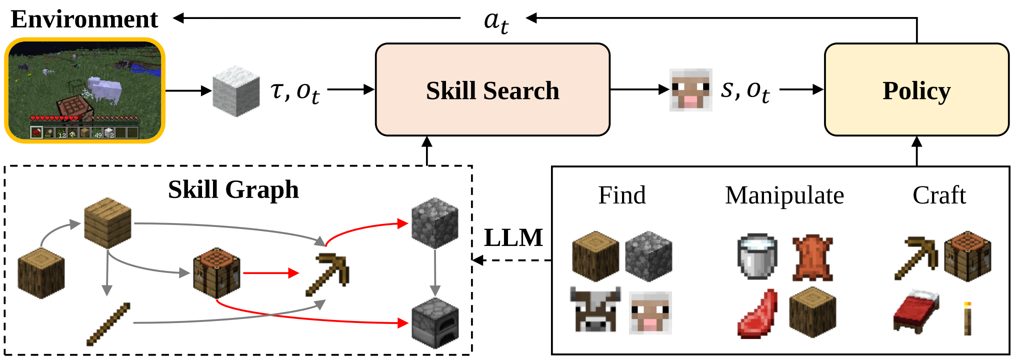
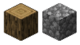
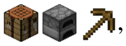


Figure 1: Overview of **Plan4MC**. We categorize the basic skills in Minecraft into three types: Finding-skills, Manipulation-skills and Crafting-skills. We train policies to acquire skills with reinforcement learning. With the help of LLM, we extract relationships between skills and construct a skill graph in advance, as shown in the dashed border. During online planning, the skill search algorithm walks on the pre-generated graph, decomposes the task into an executable skill sequence, and interactively selects policies to solve complicated tasks.

Unlike the previous definition of skills [31], we define more fine-grained basic skills and classify them into three types. In our method, each basic skill solves an atomic task that may not be further divided. Such tasks have a shorter horizon and require exploration in smaller regions of the world. Thus, using RL to learn these basic skills is more feasible. To improve the sample efficiency of RL, we introduce intrinsic rewards to train policies for different types of skills.

For high-level skill planning, recent works [3, 31] demonstrate promising results via interacting with Large Language Models (LLMs). Though LLMs generalize to open-ended environments well and produce reasonable skill sequences, fixing their uncontrollable mistakes requires careful prompt engineering [14, 31]. To make more flawless skill plans, we propose a complementary skill search approach. In the preprocessing stage, we use an LLM to generate the relationships between skills and construct a skill dependency graph. Then, given any task and the agent's condition (e.g., available resources/tools), we propose a search algorithm to interactively plan for the skill sequence. Figure 1 illustrates our proposed framework, **Plan4MC**.

In experiments, we build 24 diverse tasks in the MineDojo [8] simulator. These tasks involve executing diverse skills, including collecting basic materials , crafting useful items , and interacting with mobs . Each task requires planning and execution for 2~30 basic skills and takes thousands of environmental steps. Results show that our method accomplishes all the tasks and outperforms the baselines significantly.

To summarize, our main contributions are:

- To enable RL methods to efficiently solve diverse tasks in Minecraft, we propose three types of fine-grained basic skills and train policies to learn basic skills with intrinsic rewards. Thus, solving diverse tasks is transformed into planning over basic skills.
- Unlike previous planning methods, we propose the skill graph and the skill search algorithm for interactive planning. The LLM generates skill relationships in advance, which can be easily checked and corrected manually, to avoid uncontrollable mistakes of the LLM.
- Our hierarchical agent achieves promising performance in difficult and diverse Minecraft tasks, demonstrating the great potential of using RL to build multi-task agents in open-ended worlds.

2 Preliminaries

2.1 Problem Formulation

In Minecraft, a task $\tau = (g, I)$ is defined with the combination of a goal g and the agent's initial condition I, where g represents the target entity to acquire in the task and I represents the initial tools and conditions provided for the agent. For example, a task can be 'harvest cooked_beef with sword in plains'.

To solve complicated tasks, humans acquire and reuse skills in the world, rather than learn each task independently. Similarly, to solve the above task, the agent can sequentially use the skills: harvest log , ..., craft furnace , harvest beef , place furnace , and craft cooked_beef . Each skill solves a simple sub-task in a shorter time horizon, with the necessary tools and conditions provided. For example, the skill 'craft cooked_beef ' solves the task 'harvest cooked_beef with beef , log , and placed furnace '. Once the agent acquires an abundant set of skills S, it can solve any complicated task by decomposing it into a sequence of sub-tasks and executing the skills in order. Meanwhile, by reusing a skill to solve different tasks, the agent is much better in memory and learning efficiency.

To this end, we convert the goal of solving diverse tasks in Minecraft into building a hierarchical agent. At the low level, we train policies π_s to learn all the skills $s \in S$, where π_s takes as input the RGB image and some auxiliary information (compass, location, biome, etc.), then outputs an action. At the high level, we study planning methods to convert a task τ into a skill sequence $(s_{\tau,1}, s_{\tau,2}, \cdots)$.

2.2 Skills in Minecraft

Recent works mainly rely on imitation learning to learn Minecraft skills efficiently. In MineRL competition [15], a human gameplay dataset is accessible along with the Minecraft environment. All of the top methods in competition use imitation learning to some degree, to learn useful behaviors in limited interactions. In VPT [2], a large policy model is pre-trained on a massive labeled dataset using behavior cloning. By fine-tuning on smaller datasets, policies are acquired for diverse skills.

However, without demonstration datasets, learning Minecraft skills with reinforcement learning (RL) is difficult. MineAgent [8] shows that PPO [27] can only learn a small set of skills. PPO with sparse reward fails in 'milk a cow' and 'shear a sheep', though the distance between target mobs and the agent is set within 10 blocks. We argue that with the high dimensional state and action space, open-ended large world, and partial observation, exploration in Minecraft tasks is extremely difficult.

We conduct a study for RL to learn skills with different difficulties in Table 1. We observe that RL has comparable performance to imitation learning only when the task-relevant entities are initialized very close to the agent. Otherwise, RL performance decreases significantly. This motivates us to further divide skills into fine-grained skills. For example, the skill of 'milk a cow' becomes 'find a cow' and 'harvest milk_bucket'. After finding a cow nearby, 'harvest milk_bucket' can be accomplished by RL with acceptable sample efficiency. Thus, learning such fine-grained skills is easier for RL, and they together can still accomplish the original task.

Table 1: Minecraft skill performance of imitation learning (behavior cloning with MineCLIP backbone, reported in [4]) versus reinforcement learning. *Better init.* means target entities are closer to the agent at initialization. The RL method for each task is trained with proper intrinsic rewards. All RL results are averaged on the last 100 training epochs and 3 training seeds.

Skill					
Behavior Cloning	–	–	0.25	0.27	0.16
RL	0.40±0.20	0.26±0.22	0.04±0.02	0.04±0.01	0.00±0.00
RL (*better init.*)	0.99±0.01	0.81±0.02	0.16±0.06	0.14±0.07	0.44±0.10

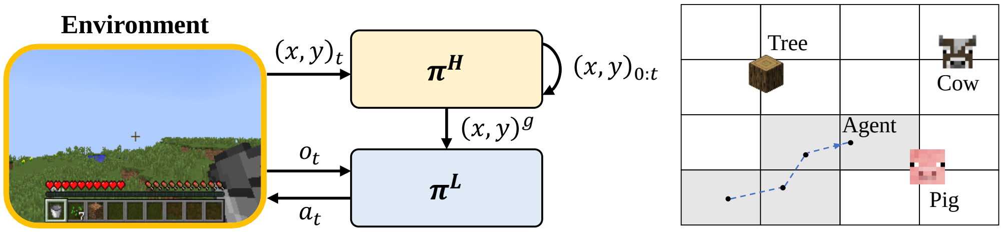


Figure 2: The proposed hierarchical policy for Finding-skills. The high-level recurrent policy π^H observes historical positions $(x, y)_{0:t}$ from the environment and generates a goal position $(x, y)^g$. The low-level policy π^L is a goal-based policy to reach the goal position. The right figure shows a top view of the agent's exploration trajectory, where the walking paths of the low-level policy are shown in blue dotted lines, and the goal is changed by the high-level policy at each black spot. The high-level policy is optimized to maximize the state count in the grid world, which is shown in the grey background.

3 Learning Basic Skills with Reinforcement Learning

We propose three types of fine-grained basic skills, which can compose all Minecraft tasks.

- Finding-skills: starts from any location, the agent explores to find a target and approaches the target. The target can be any block or entity that exists in the world.
- Manipulation-skills: given proper tools and the target in sight, the agent interacts with the target to obtain materials. These skills include diverse behaviors, like mining ores, killing mobs, and placing blocks.
- Crafting-skills: with requisite materials in the inventory and crafting table or furnace placed nearby, the agent crafts advanced materials or tools.

Except for crafting-skills that can be executed with only a single action, we study learning basic skills with RL.

3.1 Learning to Find with a Hierarchical Policy

Finding items is a long-horizon difficult task for RL. To find an unseen tree on the plains, the agent should take thousands of steps to cover the world map as much as possible. Also, it is too costly to train different policies for various target items. To simplify this problem, considering to explore on the world's surface only, we propose to train a target-free hierarchical policy to solve all the Finding-skills.

Figure 2 demonstrates the hierarchical policy for Finding-skills. The high-level policy $\pi^H\left((x, y)^g|(x, y)_{0:t}\right)$ observes historical locations $(x, y)_{0:t}$ of the agent, and outputs a goal location $(x, y)^g$. It drives the low-level policy $\pi^L\left(a_t|o_t, (x, y)^g\right)$ to reach the goal location. We assume that target items are uniformly distributed on the world's surface. To maximize the chance to find diverse targets, the objective for the high-level policy is to maximize its reached area. We discretize each 10×10 area in the world into a grid and reward the high-level policy with state counts in the grids. The low-level policy obtains the environmental observation o_t and the goal location $(x, y)^g$ proposed by the high-level policy, and outputs an action a_t. We reward the low-level policy with the distance change to the goal location.

To train the hierarchical policy with acceptable sample complexity, we pre-train the low-level policy with randomly generated goal locations using DQN [22], then train the high-level policy using PPO [27] with the fixed low-level policy. During test, to find a specific item, the agent first explores the world with the hierarchical policy until a target item is detected in its lidar observations. Then, the agent executes the low-level policy conditioned on the detected target's location, to reach the target item. Though we use additional lidar information here, we believe that without this information, we can also implement the success detector for Finding-skills with computer vision models [7].

3.2 Manipulation Skills

Manipulation tasks are instantiated with requisite tools in the inventory and target items nearby. For example, for the skill ‘harvest milk_bucket ’, we initialize the agent with an empty bucket and a cow nearby. For ‘harvest log ’, we spawn the agent in the forest to make sure nearby trees exist. Thus, Manipulation-skills can be accomplished in fewer environmental steps.

We adopt MineCLIP [8] to guide the agent with intrinsic rewards. The pre-trained MineCLIP model computes the CLIP reward based on the similarity between environmental observations (frames) and skill prompts. We train the agent using PPO with self-imitation learning, to maximize a weighted sum of CLIP return and extrinsic success (sparse) reward. Still, the CLIP reward may not be efficient for learning some complicated skills. In combat, we introduce distance reward and attack reward to further encourage the agent to chase and attack the mobs. In mining, we introduce distance reward to keep the agent close to the target blocks.

Details for training basic skills can be found in Appendix B.

4 Solving Minecraft Tasks via Skill Planning

In this section, we present our skill planning method for solving diverse hard tasks. A skill graph is generated in advance with a Large Language Model (LLM), enabling searching for correct skill sequences on the fly.

4.1 Constructing Skill Graph with Large Language Models

A correct plan $(s_{\tau,1}, s_{\tau,2}, \cdots)$ for a task $\tau = (g, I)$ should satisfy two conditions. (1) For each i, $s_{\tau,i}$ is executable after $(s_{\tau,1}, \cdots, s_{\tau,i-1})$ are accomplished sequentially with initial condition I. (2) The target item g is obtained after all the skills are accomplished sequentially, given initial condition I. To enable searching for such plans, we should be able to verify whether a plan is correct. Thus, we should know what condition is required and what is obtained for each skill. We define such information of skills in a structured format. As an example, information for skill ‘crafting stone_pickaxe ’ is:

```
stone_pickaxe {consume: {cobblestone: 3, stick: 2},
require: {crafting_table_nearby: 1}, obtain: {stone_pickaxe: 1}}
```

Each item in this format is also a skill. Regarding them as graph nodes, this format shows a graph structure between skill ‘stone_pickaxe’ and skills ‘cobblestone’, ‘stick’, ‘crafting_table_nearby’. The directed edge from ‘cobblestone‘ to ‘stone_pickaxe’ is represented as (3, 1, consume), showing the quantity relationship between parent and child, and that the parent item will be consumed during skill execution. In fact, in this format, all the basic skills in Minecraft construct a large directed acyclic graph with hundreds of nodes. The dashed border in Figure 1 shows a small part of this graph, where grey arrows denote ‘consume’ and red arrows denote ‘require’.

To construct the skill graph, we generate structured information for all the skills by interacting with ChatGPT [24], a high-performance LLM. Since LLMs are trained on large-scale internet datasets, they obtain rich knowledge in the popular game Minecraft. In prompt, we give a few demonstrations and explanations about the format, then ask ChatGPT to generate other skills information. We find that ChatGPT makes few mistakes, and these mistakes can be easily detected and fixed by human prior. Dialog with ChatGPT can be found in Appendix C.

4.2 Skill Search Algorithm

Our skill planning method is a depth-first search (DFS) algorithm on the skill graph. Given a task $\tau = (g, I)$, we start from the node g and do DFS toward its parents, opposite to the edge directions. In this process, we maintain all the possessing items starting from I. Once conditions for the skill are satisfied or the skill node has no parent, we append this skill into the planned skill list and modify the maintained items according to the skill information. The resulting skill list is ensured to be executable and target-reaching. Algorithm 2 presents the pseudocode for the skill search algorithm.

Since the learned low-level skills are possible to fail, we alternate skill planning and skill execution to solve a task, until the episode terminates. After each skill execution, we update the agent's condition I' based on its inventory and the last executed skill, and search for the next skill with $\tau' = (g, I')$. We summarize this testing process in Algorithm 3.

5 Experiments

In this section, we evaluate and analyze our method with baselines and ablations in challenging Minecraft tasks. Section 5.1 introduces the implementation of basic skills. In Section 5.2, we introduce the setup for our evaluation task suite. In Section 5.3 and 5.4, we present the experimental results and analyze skill learning and planning respectively.

5.1 Training Basic Skills

To pre-train basic skills with RL, we use the environments of programmatic tasks in MineDojo [8]. We initialize target mobs or resources closer to the agent in some environments, to mitigate the exploration difficulty. For Manipulation-skills and the low-level policy of Finding-skills, we adopt the policy architecture of MineAgent [8], which uses a fixed pre-trained CLIP image encoder and processes features using MLPs. To explore in a compact action space, we compress the original large action space into 12 × 3 discrete actions. For the high-level policy of Finding-skills, which observes the agent's past locations, we use an LSTM policy and train it with truncated BPTT [25]. We pick the best model according to the training curve for each skill, and fix these policies in all tasks. Implementation details can be found in Appendix B.

5.2 Task Setup

Based on MineDojo [8] programmatic tasks, we set up an evaluation benchmark consisting of 24 difficult tasks, including three main types: 7 tasks about cutting trees to craft primary items, 7 tasks about mining cobblestones to craft advanced items, and 10 tasks about interacting with mobs to harvest food and materials. With our settings of basic skills, these tasks require 11 planning steps on average and maximally 30 planning steps. We estimate the number of the required steps for each task with the sum of the steps of the initially planned skills and double this number to be the maximum episode length for the task, allowing skill executions to fail. The easiest tasks have 3000 maximum steps, while the hardest tasks have 10000. More details about task setup are listed in Appendix D. To evaluate the success rate on each task, we average the results over 30 test episodes.

5.3 Skill Learning

We first analyze learning basic skills. While we propose three types of fine-grained basic skills, others directly learn more complicated and long-horizon skills. We introduce two baselines to study learning skills with RL.

MineAgent [8]. Without decomposing tasks into basic skills, MineAgent solves tasks using PPO and self-imitation learning with the CLIP reward. For fairness, we train MineAgent in the test environment for each task. The training takes 6.6M environmental steps, which is equal to the sum of environmental steps we take for training all the basic skills. We average the success rate of trajectories in the last 100 training epochs (around 1M environment steps) to be its test success rate. Since MineAgent has no actions for crafting items, we hardcode the crafting actions into the training code. During trajectory collection, at each time step where the skill search algorithm returns a Crafting-skill, the corresponding crafting action will be executed. Note that, if we expand the action space for MineAgent rather than automatically execute crafting actions, the exploration will be much harder.

Plan4MC w/o Find-skill. None of the previous work decomposes a skill into executing Finding-skills and Manipulation-skills. Instead, finding items and manipulations are done with a single skill. Plan4MC w/o Find-skill implements such a method. It skips all the Finding-skills in the skill plans during test. Manipulation-skills take over the whole process of finding items and manipulating them.

Table 2 shows the test results for these methods. Plan4MC largely outperforms two baselines on the three task sets. MineAgent fails on the task sets of Cut-Trees and Mine-Stones, since taking

Table 2: Average success rates on three tasks subsets of our method, all the baselines and ablation methods. Success rates on all the single tasks are listed in Appendix E.

Task Suite	Cut-Trees	Mine-Stones	Interact-Mobs
MineAgent	0.005	0.036	0.171
Plan4MC w/o Find-skill	0.133	0.129	0.170
Interactive LLM	0.110	0.052	0.247
Plan4MC Zero-shot	0.167	0.000	0.133
Plan4MC 1/2-steps	0.286	0.205	0.277
Plan4MC	**0.376**	**0.367**	**0.320**

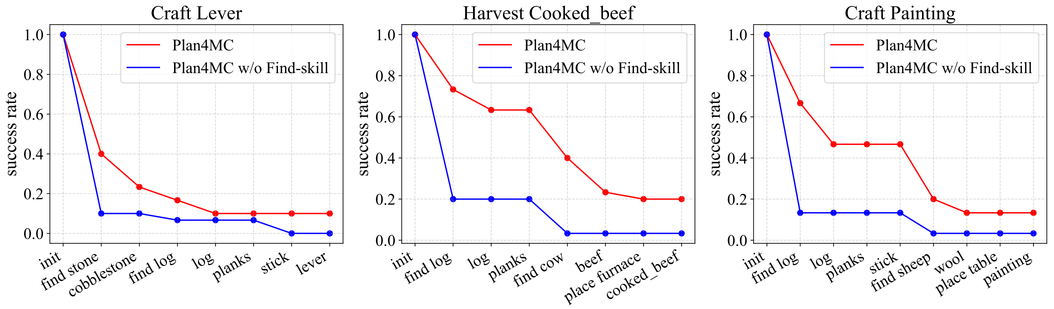


Figure 3: Success rates of Plan4MC with/without Finding-skills at each skill planning step, on three long-horizon tasks. We arrange the initially planned skill sequence on the horizontal axis and remove the repeated skills. The success rate of each skill represents the probability of successfully executing this skill at least once in a test episode. Specifically, the success rate is always 1 at task initialization, and the success rate of the last skill is equal to the task's success rate.

many attacking actions continually to mine a block in Minecraft is an exploration difficulty for RL on long-horizon tasks. On the contrary, MineAgent achieves performance comparable to Plan4MC's on some easier tasks in Interact-Mobs, which requires fewer environmental steps and planning steps. Plan4MC w/o Find-skill consistently underperforms Plan4MC on all the tasks, showing that introducing Finding-skills is beneficial for solving hard tasks with basic skills trained by RL.

To further study Finding-skills, we present the success rate at each planning step in Figure 3 for three tasks. The curves of Plan4MC and Plan4MC w/o Find-skill have large drops at Finding-skills. Especially, the success rates at finding cobblestones and logs decrease the most, because these items are harder to find in the environment compared to mobs. In these tasks, we compute the average success rate of Manipulation-Skills, conditioned on the skill before the last Finding-skills being accomplished. While Plan4MC has a conditional success rate of 0.40, Plan4MC w/o Find-skill decreases to 0.25, showing that solving sub-tasks with additional Finding-skills is more effective.

As shown in Table 3, most Manipulation-skills have slightly lower success rates in test than in training, due to the domain gap between test and training environments. However, this decrease does not occur in skills that are trained with a large initial distance of mobs/items, as pre-executed Finding-skills provide better initialization for Manipulation-skills during the test and thus the success rate may increase. In contrast, the success rates in the test without Finding-skills are significantly lower.

5.4 Skill Planning

For skill planning in open-ended worlds, recent works [12, 14, 3, 18, 31] generate plans or sub-tasks with LLMs. We study these methods on our task set and implement a best-performing baseline to compare with Plan4MC.

Table 3: Success rates of Manipulation-skills in training and test. *Training init. distance* is the maximum distance for mobs/items initialization in training skills. Note that in test, executing Finding-skills will reach the target items within a distance of 3. *Training success rate* is averaged over 100 training epochs around the selected model's epoch. *Test success rate* is computed from the test rollouts of all the tasks, while *w/o Find* refers to Plan4MC w/o Find-skill.

Manipulation-skills	Place						
Training init. distance	--	10	10	2	2	--	--
Training success rate	0.98	0.50	0.27	0.21	0.30	0.56	0.47
Test success rate	0.77	0.71	0.26	0.27	0.16	0.33	0.26
Test success rate (w/o Find)	0.79	0.07	0.03	0.03	0.02	0.05	0.06

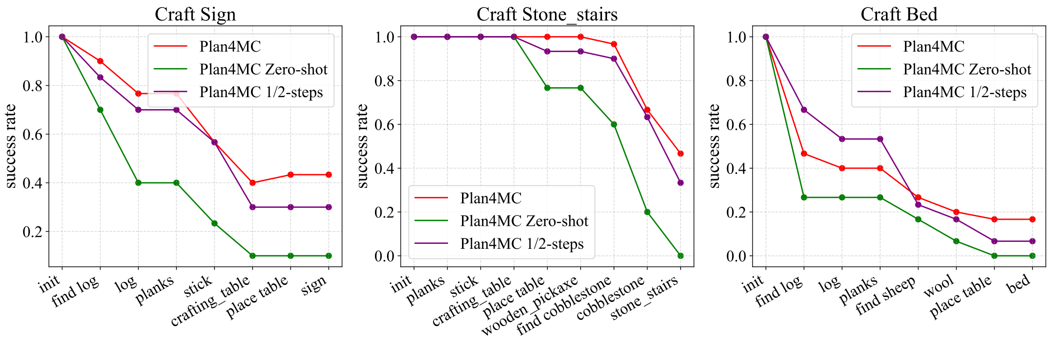


Figure 4: Success rates of Plan4MC compared with ablated skill planning methods at each planning step, on three long-horizon tasks. We arrange the initial planned skill sequence on the horizontal axis and remove the repeated skills. The success rate of each skill represents the probability of successfully executing this skill at least once in a test episode. Specifically, the success rate is always 1 at task initialization, and the success rate of the last skill is equal to the task's success rate.

Interactive LLM. We implement an interactive planning baseline using LLMs. We take ChatGPT [24] as the planner, which proposes skill plans based on prompts including descriptions of tasks and observations. Similar to chain-of-thoughts prompting [32], we provide few-shot demonstrations with explanations to the planner at the initial planning step. In addition, we add several rules for planning into the prompt to fix common errors that the model encountered during test. At each subsequent planning step, the planner will encounter one of the following cases: the proposed skill name is invalid, the skill is already done, skill execution succeeds, and skill execution fails. We carefully design language feedback for each case and ask the planner to re-plan based on inventory changes.

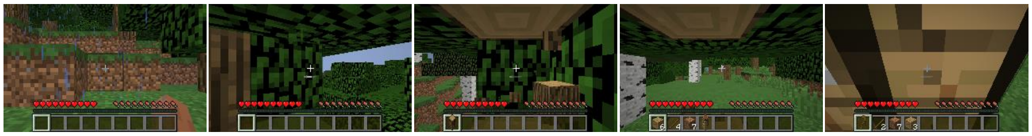
Also, we conduct ablations on our skill planning designs.

Plan4MC Zero-shot. This is a zero-shot variant of our interactive planning method, proposing a skill sequence at the beginning of each task only. The agent executes the planned skills sequentially until a skill fails or the environment terminates. This planner has no fault tolerance for skills execution.

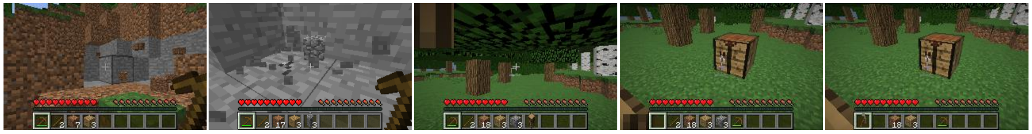
Plan4MC 1/2-steps. In this ablation study, we half the test episode length and require the agent to solve tasks more efficiently.

Success rates for each method are listed in Table 2. We find that Interactive LLM has comparable performance to Plan4MC on the task set of Interact-Mobs, where most tasks require less than 10 planning steps. In Mine-Stones tasks with long-horizon planning, the LLM planner is more likely to make mistakes, resulting in worse performance. The performance of Plan4MC Zero-shot is much worse than Plan4MC in all the tasks, since a success test episode requires accomplishing each skill in one trial. The decrease is related to the number of planning steps and skills success rates in Table 3.

As shown in Figure 4, Plan4MC 1/2-steps has close performance to Plan4MC at each planning step, while the Plan4MC Zero-shot fails after executing skills with low success rates. Plan4MC 1/2-steps has the least performance decrease to Plan4MC, showing that Plan4MC can solve tasks in a very limited episode length.



Init | Find trees | Harvest logs | Craft planks, sticks, table | Craft wooden pickaxe



Find cobblestones | Harvest cobblestones | Harvest more logs | Craft and place table | Craft stone pickaxe

Figure 5: Snapshots of a test episode for crafting stone pickaxe with bare hands.

5.5 Obtain Stone Pickaxe with Bare Hands

Obtaining stone pickaxe with bare hands is the *most* challenging task in Minecraft tech tree with our basic skills, which involves 25 planning steps and 10 basic skills. Note that this task is not in the 24 benchmark tasks. Plan4MC achieves a success rate of 18% in 100 trials, while all other methods fail with 0% success rates. A test episode is shown in Figure 5.

6 Related Work

Minecraft. In recent years, the open-ended world Minecraft has received wide attention in machine learning research. MineRL [9] and MineDojo [8] build benchmark environments and datasets for Minecraft. Previous works in MineRL competition [21, 10, 15] study the ObtainDiamond task with hierarchical RL [21, 28, 20, 19] and imitation learning [1, 10]. Other works explore multi-task learning [30, 16, 4], unsupervised skill discovery [23], planning [31], and pre-training from videos [2, 8, 6]. Our work falls under reinforcement learning and planning in Minecraft.

Learning Skills in Minecraft. Acquiring skills is crucial for solving long-horizon tasks in Minecraft. Hierarchical approaches [20, 19] in MineRL competition learn low-level skills with imitation learning. VPT [2] labels internet-scale datasets and pre-trains a behavior-cloning agent to initialize for diverse tasks. Recent works [4, 31] learn skills based on VPT. Without expert demonstrations, MineAgent [8] and CLIP4MC [6] learn skills with RL and vision-language rewards. But they can only acquire a small set of skills. Unsupervised skill discovery [23] learns skills that only produce different navigation behaviors. In our work, to enable RL to acquire diverse skills, we learn fine-grained basic skills with intrinsic rewards.

Planning with Large Language Models. With the rapid progress of LLMs [24, 5], many works study LLMs as planners in open-ended worlds. To ground language models, SayCan [3] combines LLMs with skill affordances to produce feasible plans, Translation LMs [13] selects demonstrations to prompt LLMs, and LID [17] finetunes language models with tokenized interaction data. Other works study interactive planning for error correction. Inner Monologue [14] proposes environment feedback to the planner. DEPS [31] introduces descriptor, explainer, and selector to generate plans by LLMs. In our work, we prompt the LLM to generate a skill graph and introduce a skill search algorithm to eliminate planning mistakes.

7 Conclusion

In this paper, we study solving diverse tasks in Minecraft with reinforcement learning. To tackle the exploration and sample efficiency issues, we propose Plan4MC to learn basic skills with reinforcement learning and plan for tasks with the skill search algorithm on the skill graph. Experiments on 24 challenging Minecraft tasks verify the advantages of Plan4MC over various baselines. A limitation of this work is that Finding-skills cannot explore the underground world. Future work needs to improve this skill and extend our method to more tasks.

References

[1] Artemij Amiranashvili, Nicolai Dorka, Wolfram Burgard, Vladlen Koltun, and Thomas Brox. Scaling imitation learning in Minecraft. *arXiv preprint arXiv:2007.02701*, 2020.

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

[3] 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 (CORL)*, 2023.

[4] Shaofei Cai, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. *arXiv preprint arXiv:2301.10034*, 2023.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[6] Ziluo Ding, Hao Luo, Ke Li, Junpeng Yue, Tiejun Huang, and Zongqing Lu. CLIP4MC: An rl-friendly vision-language model for Minecraft. *arXiv preprint arXiv:2303.10571*, 2023.

[7] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.

[8] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[9] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. *Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[10] William Hebgen Guss, Stephanie Milani, Nicholay Topin, Brandon Houghton, Sharada Mohanty, Andrew Melnik, Augustin Harter, Benoit Buschmaas, Bjarne Jaster, Christoph Berganski, et al. Towards robust and domain agnostic reinforcement learning competitions: MineRL 2020. In *NeurIPS 2020 Competition and Demonstration Track*, 2021.

[11] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[12] 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 (ICML)*, 2022.

[13] 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 (ICML)*, 2022.

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[15] Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, et al. MineRL diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS 2021 Competitions and Demonstrations Track*, 2022.

[16] Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint arXiv:2106.14876*, 2021.

[17] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[18] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[19] Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, and Wei Yang. Juewu-mc: Playing Minecraft with sample-efficient hierarchical reinforcement learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

[20] Hangyu Mao, Chao Wang, Xiaotian Hao, Yihuan Mao, Yiming Lu, Chengjie Wu, Jianye Hao, Dong Li, and Pingzhong Tang. Seihai: A sample-efficient hierarchical ai for the MineRL competition. In *Distributed Artificial Intelligence (DAI)*, 2022.

[21] Stephanie Milani, Nicholay Topin, Brandon Houghton, William H Guss, Sharada P Mohanty, Keisuke Nakata, Oriol Vinyals, and Noboru Sean Kuno. Retrospective analysis of the 2019 MineRL competition on sample efficient reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, 2020.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[23] Juan José Nieto, Roger Creus, and Xavier Giro-i Nieto. Unsupervised skill-discovery and skill-learning in Minecraft. *arXiv preprint arXiv:2107.08398*, 2021.

[24] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[25] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory gym: Partially observable challenges to memory-based agents. In *International Conference on Learning Representations (ICLR)*, 2023.

[26] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[28] Alexey Skrynnik, Aleksey Staroverov, Ermek Aitygulov, Kirill Aksenov, Vasilii Davydov, and Aleksandr I Panov. Hierarchical deep q-network from imperfect demonstrations in Minecraft. *Cognitive Systems Research*, 65:74–78, 2021.

[29] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

[30] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in Minecraft. In *Proceedings of the AAAI conference on artificial intelligence (AAAI)*, 2017.

[31] 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. *arXiv preprint arXiv:2302.01560*, 2023.

[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[33] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CORL)*, 2020.

A Algorithms

We present our algorithm sketches for skill planning and solving hard tasks here.

```
**Algorithm 1:** DFS.
**Input:** Pre-generated skill graph: $G$; Target item: $g$; Target item quantity: $n$;
*Global variables*: possessing items $I$ and skill sequence $S$.
for $g'$ in $parents(G, g)$ do
    $n_{g'}, n_g, consume \leftarrow < g', g >$;
    $n_{g'}^{todo} \leftarrow n_{g'}$;
    if (quantity of $g'$ in $I$) $> n_{g'}$ then
        Decrease $g'$ quantity with $n_{g'}$ in $I$, if $consume$;
    else
        $n_{g'}^{todo} \leftarrow n_{g'}^{todo} -$ (quantity of $g'$ in $I$);
    while $n_{g'}^{todo} > 0$ do
        DFS($G, g', n_{g'}^{todo}, I, S$);
        if $g'$ is not Crafting-skill then
            Remove all nearby items in $I$;
        $n_{g'}^{obtain} \leftarrow$ (quantity of $g'$ obtained after executing skill $g'$);
        if $n_{g'}^{obtain} > n_{g'}^{todo}$ then
            Increase $g'$ quantity with $n_{g'}^{obtain} - n_{g'}^{todo}$ in $I$;
        Increase other obtained items after executing skill $g'$ in $I$;
        $n_{g'}^{todo} \leftarrow n_{g'}^{todo} - n_{g'}^{obtain}$;
Append skill $g$ to $S$.
```

```
**Algorithm 2:** Skill search algorithm.
**Input:** Pre-generated skill graph: $G$; Target item: $g$; Initial items: $I$.
**Output:** Skill sequence: $(s_1, s_2, ...)$.
$S' \leftarrow ()$;
$I' \leftarrow I$;
DFS($G, g, 1, I', S'$);
return $S'$.
```

```
**Algorithm 3:** Process for solving a task.
**Input:** Task: $T = (g, I)$; Pre-trained skills: $\{\pi_s\}_{s \in S}$; Pre-generated skill graph: $G$; Skill search
        algorithm: $Search$.
**Output:** Task success.
$I' \leftarrow I$;
while task not done do
    $(s_1, s_2, ...) \leftarrow Search(G, g, I')$;
    Execute $\pi_{s_1}$ for several steps;
    if task success then
        return True;
    $I' \leftarrow$ inventory items $\cup$ nearby items;
return False.
```

B Details in Training Basic Skills

Table 4 shows the environment and algorithm configurations for training basic skills. Intrinsic rewards are implemented as follows.

State count. The high-level recurrent policy for Finding-skills optimizes the visited area in a 110×110 square, where the agent's spawn location is at the center. We divide the square into 11×11 grids and keep a visitation flag for each grid. Once the agent walks into an unvisited grid, it receives $+1$ state count reward.

Goal navigation. The low-level policy for Finding-skills is encouraged to reach the goal position. The goal location is randomly sampled in 4 directions at a distance of 10 from the agent. To get closer to the goal, we compute the distance change between the goal and the agent: $r_d = -(d_t - d_{t-1})$, where d_t is the distance on the plane coordinates at time step t. Additionally, to encourage the agent to look in its walking direction, we add rewards to regularize the agent's yaw and pitch angles: $r_{yaw} = yaw \cdot g, r_{pitch} = cos(pitch)$, where g is the goal direction. The total reward is:

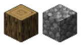
$$r = r_{yaw} + r_{pitch} + 10 * r_d. \tag{1}$$

CLIP reward. This reward encourages the agent to produce behaviors that match the task prompt. We sample 31 task prompts among all the MineDojo programmatic tasks as negative samples. The pre-trained MineCLIP [8] model computes the similarities between features of the past 16 frames and prompts. We compute the probability that the frames are most similar to the task prompt: $p = \left[\text{softmax}\left(S\left(f_v, f_l\right), \{S\left(f_v, f_{l^-}\right)\}_{l^-}\right)\right]_0$, where f_v, f_l are video features and prompt features, l is the task prompt, and l^- are negative prompts. The CLIP reward is:

$$r_{\text{CLIP}} = \max\left\{p - \frac{1}{32}, 0\right\}. \tag{2}$$

Distance. The distance reward provides dense reward signals to reach the target items. For combat tasks, the agent gets a distance reward when the distance is closer than the minimal distance in history:

$$r_{distance} = \max\left\{\min_{t'<t} d_{t'} - d_t, 0\right\}. \tag{3}$$

For mining  tasks, since the agent should stay close to the block for many time steps, we modify the distance reward to encourage keeping a small distance:

$$r_{distance} = \begin{cases} d_{t-1} - d_t, & 1.5 \le d_t \le +\infty \\ 2, & d_t < 1.5 \\ -2, & d_t = +\infty, \end{cases} \tag{4}$$

where d_t is the distance between the agent and the target item at time step t, which is detected by lidar rays in the simulator.

Attack. For combat tasks, we reward the agent for attacking the target mobs. We use the tool's durability information to detect valid attacks and use lidar rays to detect the target mob. The attack reward is:

$$r_{attack} = \begin{cases} 90, & \text{if valid attack and the target at center} \\ 1, & \text{if valid attack but the target not at center} \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

For each Manipulation-skill, we use a linear combination of intrinsic reward and extrinsic success reward to train the policy.

Table 5 shows our selected basic skill policies for downstream tasks. Since Finding-skill is trained with a target-free exploration method and has no success rate during training, we pick the model with the highest return on the smoothed training curve. For other skills, we pick the models with the highest success rates on the smoothed training curves.

Table 4: Training configurations for all the basic skills. *Max Steps* is the maximal episode length. *Training Steps* shows the environment steps cost for training each skill. *Init.* shows the maximal distance to spawn mobs at environment reset. The high-level policy and low-level policy for Finding-skills are listed in two lines.

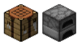
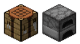
Skill	Max Steps	Method	Intrinsic Reward	Training Steps	Biome	Init.
Find	high: 40 low: 50	PPO DQN	state count goal navigation	1M 0.5M	plains	--
Place 	200	PPO	CLIP reward	0.3M	--	--
Harvest	200	PPO	CLIP reward	1M	plains	10
Harvest	200	PPO	CLIP reward	1M	plains	10
Combat	400	PPO	CLIP, distance, attack	1M	plains	2
Combat	400	PPO	CLIP, distance, attack	1M	plains	2
Harvest	500	PPO	distance	0.5M	forest	--
Harvest	1000	PPO	distance	0.3M	hills	--
Craft	1	--	--	0	--	--

Table 5: Information for all the selected basic skill policies. *Success Rate* is the success rate of the selected policy on the smoothed training curve.

Skill	Parameters	Execute Steps	Success Rate
Find	0.9M	1000	--
Place 	2.0M	200	0.98
Harvest	2.0M	200	0.50
Harvest	2.0M	200	0.27
Combat	2.0M	400	0.21
Combat	2.0M	400	0.30
Harvest	2.0M	500	0.56
Harvest	2.0M	200	0.47
Craft	0	1	1.00

C LLM Prompt Design

To construct the skill graph, we use ChatGPT to generate skill information showing relationships between skills. For both Manipulation-skills and Crafting-skills, we demonstrate several skills, explain the format, then ask ChatGPT to generate all the skills in the same format.

Listing 1: Prompt for generating some Manipulation-skills and ChatGPT's response. We only modify the last querying sentence in the prompt to generate all the skills.

```
Prompt:
I am playing the game Minecraft. I define some basic skills, like
    attack something, collect something and place something nearby. I
    list the skills in a special format.
As an example:
furnace_nearby:
  consume:
    'furnace': 1
  require:
  equip: ['furnace']
  obtain:
    'furnace_nearby': 1
```

```
To understand this skill line by line: the skill is to get a
    furnace_nearby. 'consume' means things will be consumed or killed.
     In this skill, furnace*1 will be consumed.  'require' means
    things are needed but will not be consumed. In this skill, nothing
     else is required. We should equip furnace to the first slot. If
    you do not have to equip anything, write 'equip: []'. Finally, we
    will obtain furnace_nearby*1.
Now you understand the rule of this format. Please help me generate
    the following skills:
crafting_table_nearby, wool, log, beef

ChatGPT:
Sure, here are the skills you requested in the format you provided:

crafting_table_nearby:
consume:
'crafting_table': 1
require:
equip: ['crafting_table']
obtain:
'crafting_table_nearby': 1

wool:
consume:
'sheep_nearby': 1
require:
equip: ['shears']
obtain:
'wool': 1

log:
consume:
'tree': 1
require:
equip: ['axe']
obtain:
'log': 1

beef:
consume:
'cow_nearby': 1
require:
equip: ['sword']
obtain:
'beef': 1
```

Listing 2: Prompt for generating some Crafting-skills and ChatGPT's response. We only modify the last querying sentence in the prompt to generate all the skills.

```
Prompt:
I am playing the game Minecraft. I list all the crafting skills in
    this game in a special format.
As an example:
bed:
  consume:
    'planks': 3
    'wool': 3
  require:
    'crafting_table_nearby': 1
  equip: []
  obtain:
    'bed': 1
To understand this skill line by line: the skill is to craft a bed. We
     will consume planks*3 and wool*3. To craft bed, a
    crafting_table_nearby is necessary and must be in the 'require:'
    line.
For crafting skills the equipment is always []. Finally, we will
    obtain bed*1.
Another example:
cooked_beef:
  consume:
    'planks': 1
    'beef': 1
  require:
    'furnace_nearby': 1
  equip: []
  obtain:
    'cooked_beef': 1
Because we require a furnace_nearby. The fuel, which is always 'planks
    '*1, should be added to the 'consume' items.
For the 'require:' line, you should know whether the skill needs
    crafting_table_nearby,  furnace_nearby, or nothing. If require
    nothing, just don't write anything in this line.
Now you understand the rule of this format. Please help me generate
    the following skills:
stick, chest, furnace,item_frame

ChatGPT:
Sure, here are the crafting skills for the items you requested:

stick:
consume:
'planks': 2
require:
equip: []
obtain:
'stick': 4

chest:
consume:
'planks': 8
require:
'crafting_table_nearby': 1
equip: []
obtain:
'chest': 1

furnace:
consume:
'cobblestone': 8
require:
'crafting_table_nearby': 1
equip: []
obtain:
'furnace': 1

item_frame:
consume:
'stick': 8
'leather': 1
require:
'crafting_table_nearby': 1
equip: []
obtain:
'item_frame': 1
```

All the Finding-skills have the same format:

```
(item)_nearby:
    consume:
    require:
    equip: []
    obtain:
        (item)_nearby: 1
```

In our experiments, after generating all the skills, we manually check and correct them according to the Minecraft database. Among 37 skills, we find 6 mistakes in total.

D Task Setup

Table 6 lists settings for 24 evaluation tasks. To make sure the agent is spawned in an unseen environment in each test episode, we randomly transport the agent with a maximum distance of 500 at environment reset. For tasks involving interacting with mobs, we spawn cows and sheep with a maximum distance of 30, which is much larger than the spawning distance in training basic skills.

Table 6: Settings for all the test tasks, grouped into three classes: Cut-Trees, Mine-Stones, and Interact-Mobs. *Initial Tools* are provided in the inventory at each episode beginning. *Involved Skills* is the least number of basic skills the agent should master to accomplish the task. *Planning Steps* is the number of basic skills to be executed sequentially in the initial plans.

Task Icon	Target Name	Initial Tools	Biome	Max Steps	Involved Skills	Planning Steps
	stick	--	plains	3000	4	4
	crafting_table_nearby	--	plains	3000	5	5
	bowl	--	forest	3000	6	9
	chest	--	forest	3000	6	12
	trap_door	--	forest	3000	6	12
	sign	--	forest	3000	7	13
	wooden_pickaxe	--	forest	3000	7	13
	furnace_nearby	*10	hills	5000	9	28
	stone_stairs	*10	hills	5000	8	23
	stone_slab	*10	hills	3000	8	17
	cobblestone_wall	*10	hills	5000	8	23
	lever		forest_hills	5000	7	7
	torch	*10	hills	5000	11	30
	stone_pickaxe		forest_hills	10000	9	16
	milk_bucket	, *3	plains	3000	4	4
	wool	, *2	plains	3000	3	3
	beef		plains	3000	2	2
	mutton		plains	3000	2	2
	bed	,	plains	10000	7	11
	painting	,	plains	10000	8	9
	carpet		plains	3000	3	5
	item_frame	,	plains	10000	8	9
	cooked_beef	,	plains	10000	7	7
	cooked_mutton	,	plains	10000	7	7

E Experiment Results for All the Tasks

Table 7: Success rates in all the tasks. Each task is tested for 30 episodes, set with the same random seeds across different methods.

Task	MineAgent	Plan4MC w/o Find-skill	Interactive LLM	Plan4MC Zero-shot	Plan4MC 1/2-steps	Plan4MC
	0.00	0.03	0.30	0.27	0.30	0.30
	0.03	0.07	0.17	0.27	0.20	0.30
	0.00	0.40	0.07	0.27	0.57	0.47
	0.00	0.23	0.00	0.07	0.10	0.23
	0.00	0.07	0.03	0.20	0.27	0.37
	0.00	0.07	0.00	0.10	0.30	0.43
	0.00	0.07	0.20	0.00	0.27	0.53
	0.00	0.17	0.00	0.00	0.13	0.37
	0.00	0.30	0.20	0.00	0.33	0.47
	0.00	0.20	0.03	0.00	0.37	0.53
	0.21	0.13	0.13	0.00	0.33	0.57
	0.00	0.00	0.00	0.00	0.10	0.10
	0.05	0.10	0.00	0.00	0.17	0.37
	0.00	0.00	0.00	0.00	0.00	0.17
	0.46	0.57	0.57	0.60	0.63	0.83
	0.50	0.40	0.76	0.30	0.60	0.53
	0.33	0.23	0.43	0.10	0.27	0.43
	0.35	0.17	0.30	0.07	0.13	0.33
	0.00	0.00	0.00	0.00	0.07	0.17
	0.00	0.03	0.00	0.10	0.23	0.13
	0.06	0.27	0.37	0.10	0.50	0.37
	0.00	0.00	0.00	0.03	0.10	0.07
	0.00	0.03	0.03	0.03	0.20	0.20
	0.00	0.00	0.00	0.00	0.03	0.13