SPRING: Studying the Paper and Reasoning to Play Games

Yue Wu[1,4]*, **Shrimai Prabhumoye**[2], **So Yeon Min**[1], **Yonatan Bisk**[1], **Ruslan Salakhutdinov**[1],

Amos Azaria[3], **Tom Mitchell**[1], **Yuanzhi Li**[1,4]
[1]Carnegie Mellon University, [2]NVIDIA, [3]Ariel University, [4]Microsoft Research

Abstract

Open-world survival games pose significant challenges for AI algorithms due to their multi-tasking, deep exploration, and goal prioritization requirements. Despite reinforcement learning (RL) being popular for solving games, its high sample complexity limits its effectiveness in complex open-world games like Crafter or Minecraft. We propose a novel approach, SPRING, to read Crafter's original academic paper and use the knowledge learned to reason and play the game through a large language model (LLM). Prompted with the LaTeX source as game context and a description of the agent's current observation, our SPRING framework employs a directed acyclic graph (DAG) with game-related questions as nodes and dependencies as edges. We identify the optimal action to take in the environment by traversing the DAG and calculating LLM responses for each node in topological order, with the LLM's answer to final node directly translating to environment actions. In our experiments, we study the quality of in-context "reasoning" induced by different forms of prompts under the setting of the Crafter environment. Our experiments suggest that LLMs, when prompted with consistent chain-of-thought, have great potential in completing sophisticated high-level trajectories. Quantitatively, SPRING with GPT-4 outperforms all state-of-the-art RL baselines, trained for 1M steps, without any training. Finally, we show the potential of Crafter as a test bed for LLMs. Code at github.com/holmeswww/SPRING

1 Introduction

Open-world survival games like Minecraft (Fan et al., 2022) and Crafter (Hafner, 2021) pose significant challenges for AI algorithms due to a combination of factors: procedural generation requires strong generalization; diverse action space requires multi-task capabilities; technology tree requires long-term planning and deep exploration; diverse and conflicting objectives requires goal prioritization. In particular, Crafter is designed for efficient simulation and fast iteration. Similar to Minecraft, Crafter features key challenges such as multi-tasking, exploration with a deep and wide tech-tree, requiring the agent to craft multiple tools and interact with multiple objects to survive in the game.

Reinforcement learning (RL) has been the go-to approach for game-based problems, with numerous successes in games like Go (Silver et al., 2017), robotics (Fu et al., 2020; Hafner et al., 2023) and various video games (Vinyals et al., 2019; Schrittwieser et al., 2020; Badia et al., 2020; Hafner et al., 2023). While RL demonstrated impressive performance, it still suffers from certain limitations, such as high sample complexity and difficulty in incorporating prior knowledge. Such drawbacks make it exceptionally challenging to apply RL to diverse and complex open-world benchmarks like Crafter (Hafner, 2021) or Minecraft (Fan et al., 2022). Addressing the benefits and drawbacks of RL is therefore crucial for achieving a sample-efficient solution.

*Work done during internship at Microsoft. For correspondence, contact ywu5@andrew.cmu.edu

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

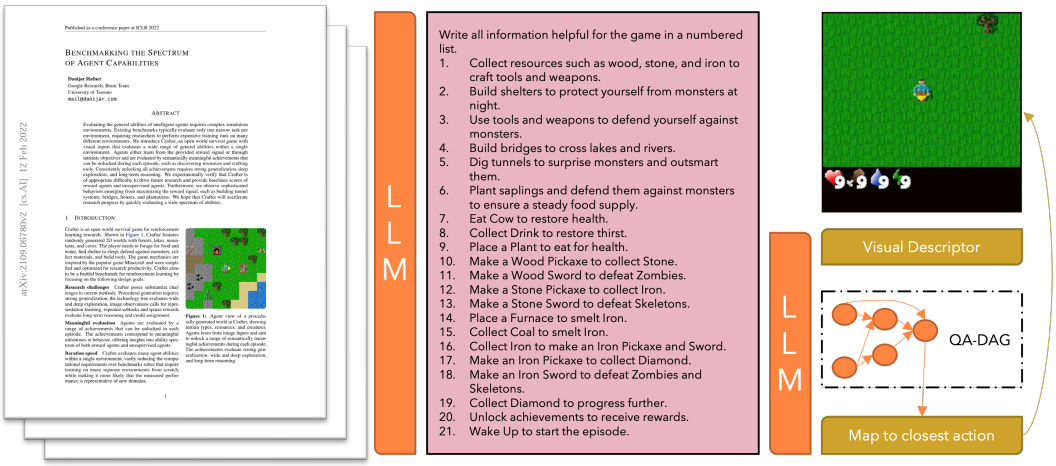

Figure 1: Overview of SPRING. The context string, shown in the middle column, is obtained by parsing the LaTeX source code of Hafner (2021). The LLM-based agent then takes input from a visual game descriptor and the context string. The agent uses questions composed into a DAG for chain-of-thought reasoning, and the last node of the DAG is parsed into action.

On the other hand, large language models (LLMs) (Brown et al., 2020; Smith et al., 2022; Chowdhery et al., 2022) have shown remarkable success when prompted for various tasks, including embodied planning and acting (Ahn et al., 2022; Du et al., 2023; Wang et al., 2023; Shinn et al., 2023), QA or dialogue (Ouyang et al., 2022; Bubeck et al., 2023), and general problem-solving (Brown et al., 2020; Bubeck et al., 2023). Their unique planning (Ahn et al., 2022), reasoning (Shinn et al., 2023), and problem-solving (Bubeck et al., 2023; Madaan et al., 2023) ability makes them a promising candidate for incorporating prior knowledge and in-context reasoning for game-based problems, particularly when it comes to addressing the aforementioned limitations of RL.

Hence, in this work, we study the possibility and reliability of LLMs for understanding and reasoning with human knowledge, in the setting of games. We consider a two staged approach SPRING (Figure 1): (1) **studying the paper**: the first stage reads the LaTeX of the paper of (Hafner, 2021) and (2) **reasoning**: the second stage involves reasoning about that knowledge through a QA framework to take an environment action. Note that the Crafter environment was released after the data collection date of GPT-3.5 and GPT 4 (OpenAI, 2023) models[2], the environment is unseen to them. We first use LLM to extract prior knowledge from the LaTeX source code of the original paper by Hafner (2021). We then use a similar QA summarization framework as Wu et al. (2023) which produces QA dialogue on game mechanics. SPRING handles significantly more diverse contextual information than (Wu et al., 2023), making use of all 17 action/interaction types and even information about desirable behaviors documented in the paper.

We focus on reading the relevant academic paper in the first stage of SPRING, by first deciding the paragraphs that are relevant for playing the game. Then we extract key information through a series of questions such as "*Write all information helpful for the game in a numbered list.*". In the second stage,we promote and regulate in-context chain-of-thought reasoning in LLMs to solve complex games. The reasoning module is a directed acyclic graph (DAG), with questions as nodes and dependencies as edges. For example, the question "*For each action, are the requirements met?*" depends on the question "*What are the top 5 actions?*", creating an edge from the latter to the former. For each environment step, we traverse the DAG computing LLM answers for each node in the topological order of the graph. The final node of the DAG is a question about the best action to take and the LLM answer for the question is directly translated to environment action.

Qualitatively, our experiments show that LLMs, when prompted with consistent chain-of-thought, can execute sophisticated trajectories independently in Crafter (Hafner, 2021). Quantitatively, SPRING's zero-shot performance with GPT-4 surpassing all state-of-the-art RL algorithmstrained for 1M steps (Table 2).

Our contributions are as follows:

- SPRING is the first to tackle a competitive RL benchmark by explicitly extracting multiple interactions and tech-tree dependencies directly from an academic paper.

[2]GPT-3.5/4 training data ends in September 2021 according to OpenAI API

- We are the first to show SOTA performance in a challenging open world game with a zero-shot LLM-based (GPT-4) policy
- We study the quality of in-context reasoning induced by different prompts and propose a controlled chain-of-thought prompting through a DAG of questions for decision making.

2 Method

This section is structured as follows. We first describe how we generate the context from the LaTeX source code of Hafner (2021) in Section 2.1. Then we describe our SPRING framework and how we compute the action in Section 2.2.

Problem Setting Our goal is to show that LLMs can plan and act reasonably well in an environment where control tasks are less required. In the setting of Crafter, we define the states, s, as samples from state distribution S. We are interested in creating a goal-conditioned policy π which maps state s to action $a \in A$, $\pi : S \rightarrow A$. Due to the use of LLM, we further break the policy down into two parts: a descriptor $\mathcal{D}$ which describes key aspects the visual observation in plain text ($d = \mathcal{D}(s)$). And an LLM-based actor which takes state description d and outputs action a.

In addition, we define $\mathcal{S}^j_{\text{para}}$ to be the j^{th} paragraph in the LaTeX source of the environment paper (Hafner, 2021), and $\mathcal{M}_{LLM}$ to be the LLM which takes a context string and a question string as input and outputs an answer to the question.

2.1 Studying the paper: Context from LaTeX source

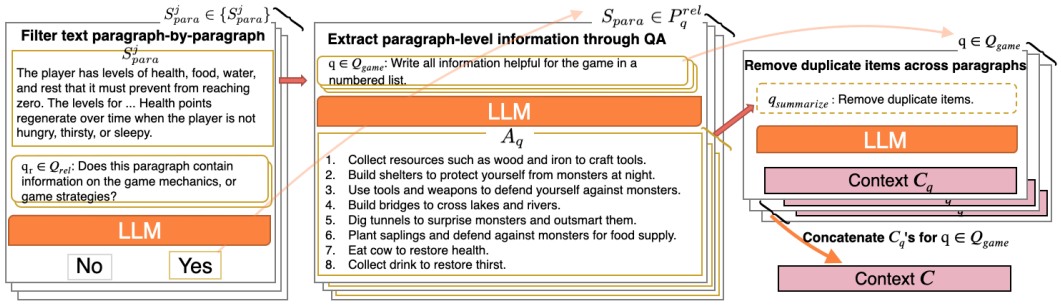

Figure 2: **Paper Studying Moudle.** The 3-step approach for computing C_q from the LaTeXsource code of Hafner (2021). First, as shown in the left column, for each paragraph we compute LLM answer for all relevancy questions in Q_{rel}, and keep only the relevant paragraphs. Second, as shown in the middle column, we compute paragraph-level LLM answer to q. Third, we summarize the answer into C_q with a summary prompt; we concatenate C_q across $q \in Q_{game}$ and obtain C.

Similar to Wu et al (2023), we compose gameplay specific questions and then compute LLM answer to the questions for each subsection in the latex files. Since a considerable amount of the paper is irrelevant to the gameplay, we use a set of 2 questions Q_{rel}={"Would this paragarph help me succeed in this game?", "Does this paragraph contain information on the game mechanics, or game strategies?"} to identify relevance, and a set of 4 questions Q_{game}={"Write all information helpful for the game in a numbered list.", "In plain text. List all objects I need to interact/avoid to survive in the game. Use "I would like to X object Y" in each step. Replace Y by the actual object, X by the actual interaction.", "Write all game objectives numbered list. For each objective, list its requirements.", "Write all actions as a numbered list. For each action, list its requirements."} to summarize gameplay and action space relevant information. We add the prompt "DO NOT answer in LaTeX." to all of Q_{game} to prevent the LLM from outputting the list in LaTeX format.

For a specific gameplay specific question $q \in Q_{\text{game}}$, our goal is to compute C_q, the answer to q conditioned on the paper. However, since the length of the paper exceeds input length constraints for most LLMs, we have to break the paper down into paragraphs individual $\mathcal{S}^j_{\text{para}}$. We provide an illustration of the process in Figure 2.

First, we filter the paragraphs for relevance and keep only paragraphs identified as relevant by at least one question from Q_{rel}. We set P^{rel}_q to be the set of relevant paragraphs.

$$P^{\text{rel}}_q = \left\{ \mathcal{S}^j_{\text{para}} | \exists q_r \in Q_{\text{rel}} \; s.t. \; \mathcal{M}_{LLM}\left(\mathcal{S}^j_{\text{para}}, q_r\right) = \text{“Yes”} \right\} \tag{1}$$

Second, we compute the set, A_q, of answers to q for each relevant paragraph from P_q^{rel}, from the LaTeX source code.

$$A_q = \left\{ \mathcal{M}_{LLM}\left(\mathcal{S}_{\text{para}}, q\right) : \mathcal{S}_{\text{para}} \in P_q^{\text{rel}} \right\} \tag{2}$$

Third, to obtain the answer string C_q from the set A_q, we query an LLM with a summarization prompt $q_{\text{summarize}}$ = "Remove duplicate items."

$$C_q = \mathcal{M}_{LLM}\left(\texttt{concat}(A_q), q_{\text{summarize}}\right) \tag{3}$$

Finally, we concatenate (with the linebreak character) all question-context pairs to form the context string C for SPRING.

$$C = \texttt{concat}\left(\{\text{"Question: } q \text{ Answer: } C_q\text{"} | \forall q \in Q_{\text{game}}\}\right) \tag{4}$$

2.2 Reasoning: QA-DAG for SPRING

Node	Question
q_1	List objects in the current observation. For each object, briefly answer what resource it provides and its requirement.
q_2	What was the last action taken by the player?
q_3	For each object in the list, are the requirements met for the interaction?
q_4	Did the last player action succeed? If not, why?
q_5	List top 3 sub-tasks the player should follow. Indicate their priority out of 5.
q_6	What are the requirements for the top sub-task? What should the player do first?
q_7	List top 5 actions the player should take and the requirement for each action. Choose ONLY from the list of all actions. Indicate their priority out of 5.
q_8	For each action in the list, are the requirements met?
q_a	Choose the best executable action from above.

Table 1: List of all 9 questions in Q_{act}. The questions are designed to promote consistent chain-of-thought. Experimentally, we find the LLM robust to different phrasing of the questions.

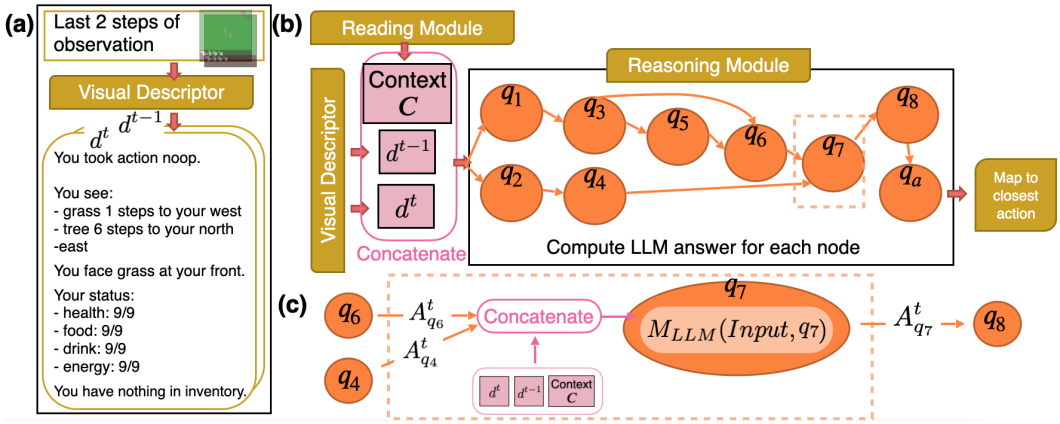

Figure 3: **Reasoning.** **(a)** The visual descriptor takes the last two gameplay screens as input, and outputs their descriptions in language (d^t, d^{t-1}). **(b)** SPRING traverses a DAG of questions from Table 1 in topological order. Answer to the final question q_a is mapped to environment action using sub-string matching. **(c)** The LLM answer for each question (node) is conditioned on the previous 2 steps of observation, the context C, and answers to the immediate parents of the current node.

For LLMs to be able to understand the gameplay, we first follow Du et al. (2023); Wang et al. (2023) to define an visual descriptor $\mathcal{M}_{\text{desc}}$ which converts state $s \in S$ to textual description d (Figure 3 a).

To achieve consistent chain-of-thought reasoning (Wei et al., 2021) throughout hundreds of steps within one round of gameplay, we compose a fixed set of questions $Q_{\text{act}} = \{q_1, \ldots, q_a\}$ to query

the LLM at every step of the game, with question-question dependencies as $D = \{(q_u, q_v) | q_u, q_v \in Q_{\text{act}}$ and answering q_v requires the answer of $q_u\}$. Note that the above specification forms a directed acyclic graph (DAG) with nodes Q_{act} edges D (Figure 3 b).

For any question (node) $q_v \in Q_{\text{act}}$, we compute the answer $A^t_{q_v}$ for time step t, conditioned on the gameplay context C, most recent 2 steps of game description d^{t-1}, d^t, and answers to its dependencies (Figure 3 c).

$$A^t_{q_v} = \mathcal{M}_{LLM}\left(\texttt{concat}\left(C, d^{t-1}, d^t, \left\{A^t_{q_u} | (q_u, q_v) \in D\right\}\right), q_v\right) \tag{5}$$

Experimentally, we find that prompting the LLM with only the direct parents of a question greatly reduces the context length, and helps LLM to focus on the most relevant contextual information.

We traverse the DAG using a modified topological sort algorithm to compute LLM answer for each question based on its topological order. Finally, we map the answer to the last question in the node q_a directly to one of the 17 named actions in the environment with sub-string matching ($a = A^t_a$). We take the default action "Do" on sub-string matching failure.[3]

3 Experiments and Results

We present our experiments as follows. First, we explain our experimental setup and baselines for our experiments. Then, we compare SPRING to popular RL methods on the Crafter benchmark. Finally, we conduct experiments and analysis on different pieces of our architecture to study the influence of each part over the in-context "reasoning" capabilities of the LLM.

3.1 Experimental Details

The Crafter environment (Hafner, 2021) is a procedurally generated open-world survival game for benchmarking RL algorithms with 22 achievements in a tech tree of depth 7. The environment is a grid-world features top-down observation and discrete action space of size 17. The observation also shows the current inventory state of the player, including its health points, food, water, rest levels, and inventory. The game is inspired by Minecraft and features a similar get-to-diamond challenge. In comparison, Crafter captures many key research challenges of Minecraft in a simpler and faster environment, thus speeding up experiments and result collection.

Environment Descriptor The gameplay screen (top left of Fig 3.) consists of a 9×9 grid ($\{(i, j) \mid 1 \leq i, j \leq 9\}$). The top 7 rows consist of the local view of the world; each cell (i, j) is associated with a pre-defined background (e.g., "grass", "water", "none") and possibly with an object "asset" (e.g., "tree", "health", "player"). The bottom 2 rows represent agent status (e.g., "health") and item inventories, which include images of assets (e.g., "stone sword"), and the number of each assent in the inventory.

Our environment descriptor accepts as input the gameplay screen and outputs a text description of the screen. We first create combinations of background and object (appearance) assets. Then we add number assets to recognize the quantity of inventory/ status. We match these combinations with the gameplay screen, using `cv2.filters` with a matching *threshold* of 0.9. We disable the detector during nights when observations are unreliable. Finally, for each (i, j), we filter the matched combinations, and select the one with the highest matching score. From this information, we can measure the distance and direction of each object relative to the player; simultaneously, we can count the agent status and inventory item.

The environment descriptor then obtains the set of objects in observation $\mathcal{O} = \{(obj, dist, direction)\}$, the set of inventory items $\mathcal{I} = \{(object, count)\}$, and the agent status $\mathcal{H} = \{(attribute, value, max)\}$. Including only the closest object of each kind, we compose the observation description d as: "You see : - <obj> <dist> steps to your <direction>. Your status: <attribute>: <value>/ <max>. Your inventory: - <object>: <count>". We describe direction of objects using "north","south","east","west".

Evaluation Metrics Agents in Crafter are evaluated primarily based on two metrics: reward and score. The game assigns a sparse $+1$ reward each time the agent unlocks a new achievement in an

[3]We will release code for our agent at github.com/anonymous

Method	Score	Reward	Training Steps[4]
Human Experts	$50.5 \pm 6.8\%$	14.3 ± 2.3	N/A
SPRING + paper (Ours)	$\mathbf{27.3 \pm 1.2\%}$	$\mathbf{12.3 \pm 0.7}$	**0**
DreamerV3 (Hafner et al., 2023)	$14.5 \pm 1.6\%$	$\mathbf{11.7 \pm 1.9}$	1M
ELLM (Du et al., 2023)	N/A	6.0 ± 0.4	5M
EDE (Jiang et al., 2022)	$11.7 \pm 1.0\%$	N/A	1M
DreamerV2 (Hafner et al., 2020)	$10.0 \pm 1.2\%$	9.0 ± 1.7	1M
PPO (Schulman et al., 2017)	$4.6 \pm 0.3\%$	4.2 ± 1.2	1M
Rainbow (Hessel et al., 2018)	$4.3 \pm 0.2\%$	5.0 ± 1.3	1M
Plan2Explore (Sekar et al., 2020)	$2.1 \pm 0.1\%$	2.1 ± 1.5	1M
RND (Burda et al., 2018)	$2.0 \pm 0.1\%$	0.7 ± 1.3	1M
Random	$1.6 \pm 0.0\%$	2.1 ± 1.3	0

Table 2: Table comparing SPRING and popular RL algorithms in terms of game score, reward, and training steps. Results for SPRING is summarized over 5 independent trials. SPRING out-performs the previous SOTA in terms of all metrics. In addition, since SPRING gathers knowledge from reading the paper, it requires no training.

episode, and assigns reward of $-0.1/0.1$ when the agent loses/gains one health point. The score metric (Hafner, 2021) is computed by aggregating the success rates for each achievement:

$$S = \exp\left(\frac{1}{N}\sum_{i=1}^{N}\ln\left(1+s_i\right)\right) - 1,$$

where s_i is the agent's success rate on achievement i and $N = 22$ is the number of achievements. Note that RL agents only train on the reward, and SPRING does not require any training.

RL Baselines We include results from popular actor-critic methods like PPO (Schulman et al., 2017); DQN variants like Rainbow (Hessel et al., 2018); intrinsically motivated methods like RND (Burda et al., 2018), Plan2Explore (Sekar et al., 2020), EDE (Jiang et al., 2022); LLM assisted solutions like ELLM Du et al. (2023); model-based methods like DreamerV2 (Hafner et al., 2020); DreamerV3 (Hafner et al., 2023), which currently holds the state-of-the-art.

LLMs. For LLM access, we use GPT-3.5-turbo (OpenAI, OpenAI) and GPT-4 (OpenAI, 2023) from OpenAI's API.

3.2 Overall Results

We compare the performance of RL baselines to SPRING with GPT-4 conditioned on the environment paper (Hafner, 2021) in Table 2.

SPRING out-performs the previous SOTA, including previous attempts at using LLMs for Crafter by large margins, achieving an 88% relative improvement on game score and a 5% improvement in reward on the best performing RL method (Hafner et al., 2023). Since the model obtains knowledge from reading the paper, SPRING requires 0 training steps, while RL methods generally require millions of training steps[4].

We include a plot of unlock rate by task, comparing our method to popular RL baselines in Figure 4. SPRING assisted by prior knowledge out-performs RL methods by more than 10x on achievements like "Make Stone Pickaxe", "Make Stone Sword", and "Collect Iron", which are up to depth 5 down in the tech tree and significantly harder to reach through random exploration. For achievements "Eat Cow" and "Collect Drink", SPRING achieves perfect performance, whereas model-based RL framework like Dreamer-V3 has more than 5x lower unlock rate for "eat cow" since cows are moving and harder to reach through random exploration. Finally, we note that SPRING did not take the action "Place Stone", which can be reached easily by random exploration, since placing a stone was not discussed as beneficial for the agent in the paper (Hafner, 2021).

[4]We base our comparison on the hard 1M cap set by Hafner (2021).

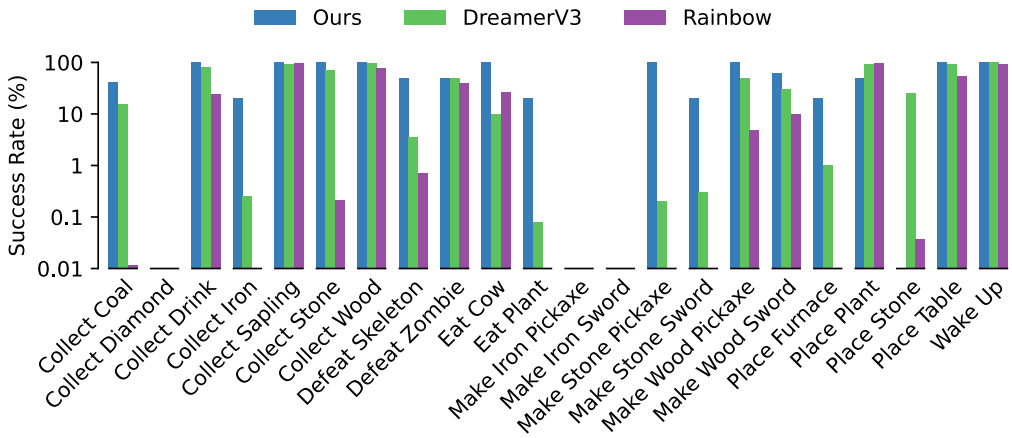

Figure 4: Ability spectrum showing the unlocking percentages for all 22 achievements. Rainbow manages to drink water and forage for food. DreamerV3 collects coal, iron, stone, and forges more advanced tools and weapons. Since SPRING starts off with knowledge about the game, it achieves more than 10x higher unlock rate on previously hard-to-reach tasks like "Eat Plant", "Make Stone Pickaxe", "Make Stone Sword", and "Collect Iron".

Method	Achievement Depth	Reward	Questions per Step
SPRING + Full Paper	6	$\mathbf{12.3 \pm 0.7}$	9
SPRING + Paper w/ modified C	4	9.4 ± 1.8	9
SPRING + Action Description	4	8.2 ± 0.2	9
SPRING + w/o C	1	0.5 ± 0.2	9
SPRING + Full Paper	6	$\mathbf{12.3 \pm 0.7}$	9
Step-by-step prompt + Full Paper	5	7.3 ± 4.4	2
QA w/o DAG + Full Paper	4	4.3 ± 3.9	9
w/o QA + Full Paper	2	2.4 ± 1.3	1
SPRING + Full Paper	6	$\mathbf{12.3 \pm 0.7}$	9
SPRING + Full Paper w/ GPT-3.5	2	3.3 ± 2.9	9

Table 3: Analysis on how different parts of SPRING contribute to its performance, comparing the max achievement depth in the tech tree, the reward, and the number of human-written questions in the prompt. Results are summarized over 5 independent trials. The first 4 rows study the necessity of prior knowledge from the context string C. The middle 4 rows study different chain-of-thought prompting techniques. The last 2 rows study the role of LLMs. All three aspects are important for SPRING to achieve best reported performance.

3.3 Component Analysis

We study how the different aspects of the framework contribute to the behavior of the agent through a series of ablations as shown in Table 3.

Studying the LaTeX Paper In the first 4 rows of Table 3, we investigate the contribution of gameplay context from the LaTeX paper toward performance of the agent. We report the performance of SPRING with no contextual information (w/o C) (row 4); SPRING conditioned on only the action descriptions and dependencies from (Hafner, 2021) Table F.1 (only question 4 from Q_{game}) (row 3); SPRING conditioned on the context manually modified to exclude the "crafting table" dependency for wooden_pickaxe by removing two corresponding lines from the context C (row 2); SPRING conditioned on the full context from the paper (row 1).

As expected, since Crafter environment is unseen for GPT, the agent achieves performance similar to random agent without any game context. When provided with only action descriptions and action dependencies, using only question 4 from Q_{game} in section 2.1, SPRING achieves strong 67% performance comparable to DreamerV2 (Silver et al., 2017).

For the next piece of the experiment, we manually remove "near crafting table" dependency for wooden_pickaxe from it's context, which is required for 11 later achievements. SPRING with GPT-4 incurs a 24% performance drop. Interestingly, we find that the LLM has some ability to recover

from the inaccurate context information. We observe that after failing to craft the wooden_pickaxe without a table, the agent instead tries to craft a wooden_sword first to maintain survival. Eventually, the agent was able to identify the missing requirement through guessing and trying after some unsuccessful trials, and craft the wooden_pickaxe. However, the confusion delayed the agent's progress and therefore causes the performance gap with the agent conditioned on the full context (row 5).

Reasoning In the middle 4 rows of Table 3, we investigate the contribution of different prompting methods toward performance of the model. Conditioned on the full context from the LaTeX paper, we report the performance of GPT-4 directly prompted to output the action using the last question q_a only (row 8); GPT-4 prompted with all questions from Q_{act} but in a list without the DAG dependencies D (row 7); GPT-4 prompted "Let's think step-by-step" (Kojima et al., 2022) about the next action, and prompted to choose a permissible action q_a with let's think step-by-step followed by q_a again (row 6); GPT-4 with SPRING (row 5).

Relative to our method, we observe that directly prompting the LLM for the action leads to a 80% performance drop, and therefore does not result in a meaningful agent. The popular chain-of-thought reasoning prompt "Let's think step-by-step" (Kojima et al., 2022) achieves reasonable reward with a 40% drop, but with a high 60.27% standard deviation. Qualitatively, we observe that the LLM produces inconsistent outputs across time steps, due to the fact that the model's chain-of-thought is not directed or controlled through the prompt. Therefore, LLMs prompted with "Let's think step-by-step" alone cannot reliably follow a good policy. Controlling the chain-of-thought with 9 questions from Q_{act} (section 2.2) successfully controls the consistency of LLM outputs across time qualitatively. However, we observe that the LLM often ignores earlier questions at later stages of QA when all previous questions are presented in a list, leading to random disagreements in answers. For example, the LLM may correctly identify that it needs "wooden pickaxe" to mine the stone ahead in the first few questions, but forgets about the requirement later when it's prompted for actions. Quantitatively, the model performs 65% worse with 90% variance without the DAG. The introduction of DAG eliminates this problem by reducing the QA context length to only a question's immediate parents.

Overall, SPRING achieves the best performance and a small 6% performance standard deviation, due to more consistent reasoning over time steps with better focus and fewer distractions.

LLM In the last two rows of Table 3, we show that the same architecture does not work well with GPT-3.5-turbo. We believe the observed 73% performance gap mainly comes from GPT-3.5-turbo's worse performance at following fine-grained instructions in each of the questions, which are required for chain-of-thought reasoning with SPRING.

3.4 Cost for running SPRING

The number of queries per step is 9 (same as the number of questions). Each game could take around 300 steps, but can go up to 500 steps in the worst case. Therefore, the maximum number of queries per game can go up to 4500. According to the public price of GPT-4 API, each query costs around 0.06[5]. The total cost should be less than 270 (USD) per game with GPT-4. Given the progress in chip-set development, we are hopeful that the inference costs will lower, making LLMs more accessible for the public.

3.5 Potential for Benchmarking LLMs

In Table 4, we compare popular publicly available LLMs including GPT-4 (OpenAI, 2023), GPT-3.5 (text-davinci-003) (OpenAI, OpenAI), Bard (Manyika, Manyika), Claude (Anthropic, Anthropic), Alpaca-30b (Taori et al., 2023) under the same setting on Crafter, following the same step-by-step prompt as Section 3.3 and Table 3. We observe a clear separation in performance under our setting.

4 Related Work

RL v.s. LLMs Comparing LLM-based agents against RL agents brings forth a intriguing discussion. RL algorithms do not require prior knowledge such as instruction manuals, and could

[5]Pricing estimated according to OpenAI API as of October 17 2023

Method	Achievement Depth	Reward	Questions per Step
Step-by-step prompt + GPT-4	5	$\mathbf{7.3 \pm 4.4}$	2
Step-by-step prompt + text-davinci-003	4	4.5 ± 2.1	2
Step-by-step prompt + Bard	0	-0.9 ± 0	2
Step-by-step prompt + Claude	1	0.1 ± 0.1	2
Step-by-step prompt + Alpaca-30b	1	0.1 ± 0.1	2
Random	1	2.1 ± 1.3	0

Table 4: Comparison of different LLMs under the same setting using the context C generated with text-davinci-003 following the same step-by-step prompt as Section 3.3 and Table 3.

continually improve given enough trials. However, RL algorithms are typically trained with reward functions deliberately engineered to cover all in-game achievements (Hafner, 2021; Hafner et al., 2023). Such reward functions often require a lot of expert knowledge and careful formulation.

On the other hand, LLM agents like SPRING does not need the reward (we report reward for comparison purpose, SPRING does not use the reward during inference), but instead uses external knowledge from the LaTeX source code. In addition, current LLM agents lack the capabilities of improving from interactions.

We hope future works would be able to leverage the benefits of both paradigms in order to achieve efficient planning with fine-grained control.

Policy Informed by Natural Language Instructions In the instruction following setting, step-by-step instructions have been used to generate auxiliary rewards, when environment rewards are sparse. Goyal et al. (2019); Wang et al. (2019) use auxiliary reward-learning modules trained offline to predict whether trajectory segments correspond to natural language annotations of expert trajectories.

There has been many attempts to go beyond instruction following to learning from unstructured natural language (Branavan et al., 2012; Goldwasser and Roth, 2014; Zhong et al., 2021; Wang and Narasimhan, 2021). Zhong et al. (2021); Wang and Narasimhan (2021) make use of special architectures to learn reasoning on grid worlds with template-generated instructions. However, the model requires 200 million training samples from templates identical to the test environments. Such a training requirement limiting the generalization of the model and causes performance loss even on slightly bigger grid worlds with identical mechanics.

Wu et al. (2023) proposes a summary (Read) and reasoning (Reward) through a QA prompting framework with an open-source QA LLM (Tafjord and Clark, 2021). The framework demonstrates the possibility of an using real-world human-written manuals to improve RL performance on popular games, despite limiting the interaction types to only "hit". Our framework handles all 17 kinds of interactions available in the game. Moreover, our framework makes use of information on tech-tree dependencies, and suggestions on desired policies extracted from the academic paper.

LLMs for Planning LLMs have shown promising results at high-level planning in indoor embodied manipulation environments. Huang et al. (2022); Ahn et al. (2022) primarily explores generating plans for embodied tasks, with limited actions space and trajectory length. Song et al. (2022); Wu et al. (2022) enhances Ahn et al. (2022) with greater action diversity and real-time re-planning. However, a lot of the high-level plans lack executability and has to be post-processed to meet specific task requirements, thus limiting the generalization to complex open world tasks. In addition, all prior works along this line operates on few-shot human/expert generated demonstrations containing up to 17 trajectories to provide context for LLMs, which requires more manual labor, and may limit the generalization to unseen scenarios. In comparison, our SPRING framework requires no demonstration.

LLMs for Open World Games Compared to popular indoor manipulation tasks, planning in open-world game environments poses the following additional challenges. 1) **Long horizon.** Due to the nature how in-game achievement/technology progresses, a successful gameplay can easily go beyond 200 steps (Hafner, 2021). 2) **Parallel objectives.** Open-world environments contain objectives that can be pursued in parallel and often require prioritization (Wang et al., 2023). Therefore, open world games are significantly more challenging than current indoor embodied manipulation environments.

Du et al. (2023) applies LLMs as high-level planners to assist RL exploration in Crafter. Wang et al. (2023); Yuan et al. (2023) use LLMs as high-level planner and goal selector to control a low level-policy in Minecraft. Tsai et al. (2023) studies the capabilities of ChatGPT on text games. Notably, all prior works require expert or human generated example trajectories as context for the LLMs. Since the example trajectories do not cover all scenarios, all prior works may encounter unseen situation during evaluation, leading to an overall performance inferior to state-of-the-art RL algorithms (Hessel et al., 2018; Guss et al., 2021; Hafner et al., 2023), trained without the use of LLMs. To our knowledge, we are the first to show an LLM (GPT-4) achieving performance surpassing the state-of-the-art RL algorithms in a challenging open world game.

5 Limitations and Future Work

A primary limitation in using an LLM to support interaction with the environment is the need for object recognition and grounding. However, these limitations do not exist in environments that offer accurate object information, such as contemporary games (Fan et al., 2022) and virtual reality worlds (Kolve et al., 2017). While pre-trained visual backbones (He et al., 2017) perform poorly on games, they have shown reasonable performance for environments closer to the real-world (Shridhar et al., 2020). In addition, with recent progress on visual-language models (Bubeck et al., 2023; Driess et al., 2023; Liu et al., 2023; Zou et al., 2023), we believe there will be reliable and generalizable solutions to visual-language understanding in the foreseeable future. Future works could focus on address the requirement for a separate visual descriptor with large visual-language models.

6 Conclusions

In this work, we explore solving the Crafter (Hafner, 2021) RL benchmark using the latest LLMs by reading the LaTeX source code of an academic paper about the benchmark. We study the quality of in-context "reasoning" and "planning" induced by different forms of prompts under the setting of the Crafter open-world environment. To enforce consistent planning and execution over hundreds of environment steps, we introduce SPRING, an innovative prompting framework for LLMs designed to enable in-context chain-of-thought planning and reasoning. Quantitatively, SPRING with GPT-4 outperforms all state-of-the-art RL baselines, trained for 1M steps, without any training.

Our work demonstrates the reliability of LLMs for understanding and reasoning with human knowledge. We hope that our work points to a new way of integrating human prior knowledge into RL training through intrinsic rewards (Wu et al., 2023), hierarchical RL (Shu et al., 2017), or sub-goal planning (Wang et al., 2023; Wu et al., 2023).

Broader Impacts

Our research on LLM holds potential for both positive and negative impacts. The benefits include better understanding of the powers of LLM and enhanced integration of prior knowledge, which could lead to advancement in various AI topics. However, the risks may involve reliance on computationally demanding models, game cheating or exploitation, and reliance on prior knowledge.

References

Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng (2022). Do as i can, not as i say: Grounding language in robotic affordances.

Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, et al. (2022). Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Anthropic. Introducing claude. https://www.anthropic.com/index/introducing-claude. Accessed: May 27, 2023.

Badia, A. P., B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell (2020). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pp. 507–517. PMLR.

Branavan, S., D. Silver, and R. Barzilay (2012). Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research 43*, 661–704.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Bubeck, S., V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Burda, Y., H. Edwards, A. Storkey, and O. Klimov (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Chowdhery, A., S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Driess, D., F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. (2023). Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

Du, Y., O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas (2023). Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*.

Fan, L., G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar (2022). Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv:2206.08853*.

Fu, J., A. Kumar, O. Nachum, G. Tucker, and S. Levine (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Goldwasser, D. and D. Roth (2014). Learning from natural instructions. *Machine learning 94*, 205–232.

Goyal, P., S. Niekum, and R. J. Mooney (2019). Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.

Guss, W. H., M. Y. Castro, S. Devlin, B. Houghton, N. S. Kuno, C. Loomis, S. Milani, S. Mohanty, K. Nakata, R. Salakhutdinov, et al. (2021). The minerl 2020 competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:2101.11071*.

Hafner, D. (2021). Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*.

Hafner, D., T. Lillicrap, M. Norouzi, and J. Ba (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

Hafner, D., J. Pasukonis, J. Ba, and T. Lillicrap (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

He, K., G. Gkioxari, P. Dollár, and R. Girshick (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

Hessel, M., J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

Huang, W., P. Abbeel, D. Pathak, and I. Mordatch (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents.

Jiang, Y., J. Z. Kolter, and R. Raileanu (2022). Uncertainty-driven exploration for generalization in reinforcement learning. In *Deep Reinforcement Learning Workshop NeurIPS 2022*.

Kojima, T., S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa (2022). Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

Kolve, E., R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi (2017). AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*.

Liu, S., Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu, et al. (2023). Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*.

Madaan, A., N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, et al. (2023). Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Manyika, J. An overview of bard: an early experiment with generative ai. https://ai.google/static/documents/google-about-bard.pdf. Accessed: May 27, 2023.

OpenAI. Gpt-3.5. https://platform.openai.com/docs/models/gpt-3-5. Accessed: May 27, 2023.

OpenAI (2023). Gpt-4 technical report.

Ouyang, L., J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems 35*, 27730–27744.

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature 588*(7839), 604–609.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sekar, R., O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak (2020). Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR.

Shinn, N., B. Labash, and A. Gopinath (2023). Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*.

Shridhar, M., X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht (2020). Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Shu, T., C. Xiong, and R. Socher (2017). Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. (2017). Mastering the game of go without human knowledge. *nature 550*(7676), 354–359.

Smith, S., M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zheng, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro (2022). Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model. *CoRR abs/2201.11990*.

Song, C. H., J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su (2022). Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088*.

Tafjord, O. and P. Clark (2021). General-purpose question-answering with macaw. *arXiv preprint arXiv:2109.02593*.

Taori, R., I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto (2023). Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca. Accessed: May 27, 2023.

Tsai, C. F., X. Zhou, S. S. Liu, J. Li, M. Yu, and H. Mei (2023). Can large language models play text games well? current state-of-the-art and open questions. *arXiv preprint arXiv:2304.02868*.

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 575*(7782), 350–354.

Wang, H. and K. Narasimhan (2021). Grounding language to entities and dynamics for generalization in reinforcement learning. *arXiv preprint arXiv:2101.07393*.

Wang, X., Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang (2019). Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6629–6638.

Wang, Z., S. Cai, A. Liu, X. Ma, and Y. Liang (2023). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Wei, J., M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le (2021). Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Wu, Y., Y. Fan, P. P. Liang, A. Azaria, Y. Li, and T. M. Mitchell (2023). Read and reap the rewards: Learning to play atari with the help of instruction manuals. *arXiv preprint arXiv:2302.04449*.

Wu, Y., S. Y. Min, Y. Bisk, R. Salakhutdinov, A. Azaria, Y. Li, T. Mitchell, and S. Prabhumoye (2023). Plan, eliminate, and track–language models are good teachers for embodied agents. *arXiv preprint arXiv:2305.02412*.

Wu, Y., S. Y. Min, Y. Bisk, R. Salakhutdinov, and S. Prabhumoye (2022). Tackling alfworld with action attention and common sense from language models.

Yuan, H., C. Zhang, H. Wang, F. Xie, P. Cai, H. Dong, and Z. Lu (2023). Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*.

Zhong, V., A. W. Hanjie, S. I. Wang, K. Narasimhan, and L. Zettlemoyer (2021). Silg: The multi-environment symbolic interactive language grounding benchmark. *arXiv preprint arXiv:2110.10661*.

Zou, X., J. Yang, H. Zhang, F. Li, L. Li, J. Gao, and Y. J. Lee (2023). Segment everything everywhere all at once. *arXiv preprint arXiv:2304.06718*.