Plancraft: an evaluation dataset for planning with LLM agents

Gautier Dagan , **Frank Keller** , **Alex Lascarides**

University of Edinburgh

{gautier.dagan, keller, alex}@ed.ac.uk

Abstract

We present Plancraft, a multi-modal evaluation dataset for LLM agents. Plancraft has both a text-only and multi-modal interface, based on the Minecraft crafting GUI. We include the Minecraft Wiki to evaluate tool use and Retrieval Augmented Generation (RAG), as well as an oracle planner and oracle RAG information extractor, to ablate the different components of a modern agent architecture. To evaluate decision-making, Plancraft also includes a subset of examples that are intentionally unsolvable, providing a realistic challenge that requires the agent not only to complete tasks but also to decide whether they are solvable at all. We benchmark both open-source and closed-source LLMs and strategies on our task and compare their performance to a handcrafted planner. We find that LLMs and VLMs struggle with the planning problems that Plancraft introduces, and we offer suggestions on how to improve their capabilities.

1 Introduction

With the increased performance and affordability of Large Language Models (LLMs) [OpenAI, 2024; Dubey *et al.*, 2024], LLM-based agents have exploded in popularity [Xi *et al.*, 2023]. We define LLM agents as systems that incorporate at least one call to an LLM, which influences its autonomous decisions about actions in an environment. LLMs are promising as backbones for agent-based systems because they can leverage general knowledge acquired during pre-training to tackle tasks expressed in natural language. This enables human interactions during planning and execution, which are challenging for other planning paradigms such as Reinforcement Learning (RL) or symbolic planners [Xi *et al.*, 2023].

LLMs can query and interpret knowledge bases, ask questions, and incorporate new information via text, all through a flexible dialogue interface. Even though they're not designed as planners, they are increasingly applied to embodied environments where various methods and strategies [Yao *et al.*, 2023b; Shinn *et al.*, 2023; Yao *et al.*, 2023a] aim to maximise the performance of autonomous agents [Xi *et al.*, 2024]. But LLMs still exhibit problematic issues that limit their reliability and usefulness for general-purpose agents – hallucinations
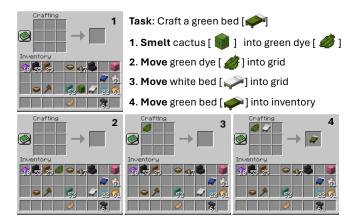

Figure 1: A Plancraft example where the task is to craft a green bed 🛏. The agent has to use the observations (text or image) to generate the next action. In this case, the shortest path is: crafting the green dye 🟢 by smelting the cactus 🌵; then moving the items (white bed 🛏 and green dye 🟢) into the correct crafting slots; then moving the resulting green bed 🛏 into the inventory.

and brittleness to inputs, limited context windows, and lack of grounding when placed in new environments.

Most existing benchmarks for evaluating LLM agents focus almost exclusively on success rates [Shridhar *et al.*, 2021; Yao *et al.*, 2022; Liu *et al.*, 2023; Xi *et al.*, 2024]: that is, the proportion of trials in which the agent achieved a goal state. Success rates measure whether the agent constructed a valid plan, but do not measure the efficiency or the quality of that plan. Some environments allow an arbitrary number of incorrect steps, while others immediately terminate an episode if a wrong action is executed. Reporting or measuring only success rates also introduces a dataset bias, as it implies that the solution to a given problem is easily verifiable, yet difficult to obtain. As a result, we argue that **success rate alone is insufficient to capture the complexity of real-world scenarios**. Each additional step of inference incurs non-negligible costs, so metrics should include more fine-grained assessments, such as how close the LLM's plan is to a handcrafted solution. Furthermore, **effective agent-based systems should recognise when a task is unsolvable**, as many real-world tasks may lie beyond the agent's capabil-

Dataset	Type	Visual Inputs	Knowledge Base	Planner	Impossible Set
Mind2Web [Deng *et al.*, 2023]	Web	✗	✗	✗	✗
WebShop [Yao *et al.*, 2022]	Web	✓	✗	✗	✗
MiniWoB++ [Liu *et al.*, 2018]	Web	✓	✗	✗	✗
WebArena [Zhou *et al.*, 2023]	Web	✗	✗	✗	✗
GAIA [Mialon *et al.*, 2023]	Web	✓	✓[†]	✗	✗
VirtualHome [Puig *et al.*, 2018]	Home	✓	✗	✗	✗
ALFWorld [Shridhar *et al.*, 2021]	Home	✗	✗	✗	✗
ALFRED [Shridhar *et al.*, 2020]	Home	✓	✗	✓	✗
MineDojo [Fan *et al.*, 2022]	Game	✓	✓	✗	✗
ScienceWorld [Wang *et al.*, 2022a]	Game	✗	✗	✗	✗
BabyAI [Chevalier-Boisvert *et al.*, 2019]	Game	✗	✗	✓	✗
TextCraft [Prasad *et al.*, 2023]	Game	✗	✗	✗	✗
Plancraft (ours)	Game	✓	✓	✓	✓

Table 1: Comparison of different interactive datasets commonly used in agentic LLM evaluation. Each of these datasets expresses the goal or task in natural language and requires multi-step planning. We design Plancraft to provide a testing ground for which there exists both a natural language knowledge base and a handcrafted planner. Plancraft also includes a subset of the data that is intentionally unsatisfiable (Impossible Set), since the capacity to predict that there's no valid plan is important. [†] GAIA allows tool use, including web search, however a knowledge base on how to solve the task is not provided.

ities and indeed many real-world tasks might be difficult to verify as doable, or not. Without the capacity to predict if there's no valid plan, an agent will incur significant costs in continually monitoring and replanning.

To this end, we introduce Plancraft, a new multi-modal planning evaluation dataset based on Minecraft, that constrains the environment to the crafting GUI (see Figure 1). Plancraft consists of planning problems of diverse complexity and length and includes a portion of the dataset that is intentionally unsolvable. Plancraft, unlike previous environments, allows us to benchmark agents against a planner, but in a setting that was designed by humans for humans. Since the crafting component of Minecraft is inherently designed for human players, Plancraft also offers a way to test how LLM agents can leverage human knowledge (in the form of the Minecraft Wiki) to solve planning tasks. We compare Plancraft with popular interactive datasets in Table 1.

We evaluate different LLM-based agents on our dataset, thereby providing LLM-agent baselines. We test both open-source and closed-source models and evaluate them against different sets of possible actions. For open-source models, we evaluate the impact of fine-tuning agents on a set of expert plans, and compare multi-modal agents to text-only models. We also fine-tune a bounding-box detection model on our environment to provide an interface through which text-only LLMs can interact with the multi-modal environment. We release all our baseline models and code along with the dataset and environment as a stand-alone Python package.[1]

2 Related Work

2.1 LLM Agents

Various strategies have been proposed to leverage LLMs as agents with interactive feedback [Yao *et al.*, 2023b; Huang *et al.*, 2022; Shinn *et al.*, 2023; Yao *et al.*, 2023a; Wang *et*

al., 2022b]. The basic idea is to harvest LLMs for information, which, if novel to the agent, may result in better behaviours than they would perform otherwise. The simplest strategy, ReAct [Yao *et al.*, 2023b], consists of interleaving actions with 'thinking' steps where the LLM is allowed unconstrained generation. Other broader strategies, such as Reflexion [Shinn *et al.*, 2023] or Inner Monologue [Huang *et al.*, 2022] try to promote self-corrective behaviour through external modules or steps. Self-Consistency [Wang *et al.*, 2022b] and Tree-of-Thought [Yao *et al.*, 2023a] sample the space of possible paths and select a solution through a majority vote or through another LLM evaluation step. Although all of these techniques are simple to implement, they can often add significant overhead in inference compute.

One limitation of LLMs is the size of their context window. To address this, modern systems often incorporate a form of Retrieval-Augmented Generation (RAG) [Lewis *et al.*, 2020], which searches a larger set of documents and then restricts the context to only relevant examples (by some quantitative metric), given the current observations and task. For example, Voyager [Wang *et al.*, 2023a] augments its context with the most likely applicable skills for a given problem. Similarly, JARVIS-1 [Wang *et al.*, 2023b] stores successful trajectories and augments the context with a relevant subset, estimated via similarity with the current observation. Since crafting is a core part of Minecraft, its Wiki contains a page with details of recipes for crafting all items. The Minecraft Wiki is therefore well suited to a RAG pipeline where an agent can query the Wiki as a knowledge source.

Enabling LLMs to use external tools, such as web search, calculators, and database queries, can also significantly expand their capabilities beyond simple next-token prediction. This has inspired numerous research efforts [Nakano *et al.*, 2022; Schick *et al.*, 2023; Parisi *et al.*, 2022; Yang *et al.*, 2023; Patil *et al.*, 2023] to integrate and evaluate tool use (or external actions) into LLMs. Some datasets, such as GAIA [Mialon *et al.*, 2023], even evaluate agents without any re-

---

[1] https://github.com/gautierdag/plancraft

striction on the tools allowed. In the context of LLM agents, actions can be seen as analogous to tools.

2.2 LLM Evaluation Datasets

Research on LLMs-based agents requires datasets and benchmarks to evaluate them. These datasets vary significantly in the types of environments in which actions are executed, ranging from simplified text-only worlds to multi-modal environments. ALFWorld [Shridhar *et al.*, 2021] and ALFRED [Shridhar *et al.*, 2020] provide virtual home environments where the agent must manipulate objects to fulfil the instruction, expressed in natural language. Mind2Web [Deng *et al.*, 2023], WebShop [Yao *et al.*, 2022], and WebArena [Zhou *et al.*, 2023] evaluate agents in web-based environments, with tasks requiring interaction with browser interfaces or e-commerce platforms. BabyAI [Chevalier-Boisvert *et al.*, 2019] is a 2D maze-solving game with natural language instructions. Minecraft is also a popular benchmark for agent evaluation, with various datasets such as MineRL [Guss *et al.*, 2019], MineDojo [Fan *et al.*, 2022], and TextCraft Prasad *et al.* [2023]. In Table 1, we compare Plancraft against popular LLM evaluation datasets. While these vary in domain and implementation, all require agents to do multi-step planning to achieve tasks expressed in natural language.

Knowledge Base Minecraft has a unique advantage in that it is a real game, rather than a construction designed for agent evaluation, and therefore there exists a variety of online content to assist human players. Several works have taken advantage of knowledge bases in Minecraft in conjunction with the game environment. Baker *et al.* [2022] collected a large dataset of Minecraft videos and trained a model, Video Pretraining (VPT), to solve tasks directly from pixels in MineRL [Guss *et al.*, 2019]. Fan *et al.* [2022] introduced MineDojo along with a knowledge base of scraped resources such as Minecraft videos and pages taken from the Minecraft Wiki and Reddit forum; however, they do not evaluate whether these improve performance. In Plancraft, we collect the pages of recipes available on the Minecraft Wiki and use them to build a knowledge base to evaluate RAG capabilities. Additionally, we implement an oracle recipe search to provide an upper estimate of a well-performing RAG system.

Planner Although all of the datasets listed in Table 1 can evaluate whether an agent achieves the goal, these lack a handcrafted policy to evaluate the quality of the agent's plans: they only measure whether the agent has reached the goal. Of the datasets that we compare to, only BabyAI [Chevalier-Boisvert *et al.*, 2019] and ALFRED [Shridhar *et al.*, 2020] release handcrafted policies or solvers. BabyAI implements a Bot Agent with handcrafted policies that can act as a teacher agent for learners in the environment. ALFRED releases expert demonstrations found using a PDDL solver and reports Path Weighted Metrics that take the length of the expert into account. Similarly, we implement a planner for Plancraft to provide a benchmark against which to compare agents.

Impossible Set All existing datasets lack a mechanism for assessing task feasibility or for handling scenarios where tasks are fundamentally unsolvable, which introduces a bias toward tasks that always assume a solution exists. Plancraft addresses this gap by including an intentional set of impossible tasks – tasks that cannot be solved under the provided constraints. By incorporating these examples, we assess an agent's capacity to reason about task feasibility. This setup promotes more realistic and cost-effective decision making, encouraging agents to balance problem solving with the practicality of task solvability.

3 Plancraft

To create Plancraft, we implement the logic and visual representation of the Minecraft crafting GUI. Similarly to Prasad *et al.* [2023], we reconstruct the crafting process entirely in Python using game files and images of items obtained from the Wiki. Using image manipulation to overlay items on top of the inventory background significantly improves performance, as the observations do not require the Java game engine. Our environment has a one-to-one pixel mapping with the real Minecraft interface and supports multiple resolutions.
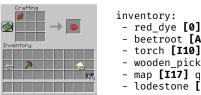
3.1 Assumptions

Plancraft makes a number of simplifying assumptions:

1. **Single Agent**: there is a single agent in the environment and any changes to the environment are a direct result of this agent's actions.

2. **Deterministic**: valid actions are always executed and their effects are predictable.

3. **Observable**: the crafting interface and inventory are fully observable, either through the text description or through the multi-modal image input.

4. **Sequential**: aligned with classical planning, each action is discrete and executed one at a time.

3.2 Action and Observation Space

The abstraction level chosen for the action and observation space has a great impact on the tractability of the planning problem. Other Minecraft environments such as MineDojo [Fan *et al.*, 2022] and Textcraft [Prasad *et al.*, 2023] provide a high-level 'craft' command that doesn't require the agent to manipulate the underlying ingredients. Plancraft, on the other hand, requires the correct placement of ingredients in the inventory and, as such, provides a lower-level symbolic action space. The two possible *environment* actions are `smelt` and `move`, and both actions require a `slot_from`, a `slot_to` and a `quantity`. This abstracts control dynamics and focuses on planning and spatial reasoning.

In the Minecraft crafting GUI, there are 46 possible item slots or positions. The first of these is the crafting slot and is only populated with an item if the correct items have been deposited within the 3×3 crafting grid. Items can only be withdrawn from the crafting slot, and moving the item out of the slot confirms the crafting operation and removes the items present in the crafting grid. The crafting grid is an arrangement of slots 3×3, where items can be moved in and out. Crafting recipes in Minecraft are either Shapeless or Shaped. Shapeless recipes only check whether the items required are present in the crafting grid, whereas Shaped recipes require the items to also be placed in the correct positions. When the

```
inventory:
 - red_dye [0] quantity 1
 - beetroot [A1] quantity 1
 - torch [I10] quantity 57
 - wooden_pickaxe [I12] quantity 1
 - map [I17] quantity 22
 - lodestone [I27] quantity 50
```

Figure 2: Example of a Plancraft observation of the scene in either an image or text format. For the text, we encode the scene using a specific slot notation where [0] denotes the crafting slot, [A1] to [C3] denotes each row and column of the 3×3 crafting grid, and [I1] to [I36] denotes the inventory slots.

items on the crafting grid match one of the 634 recipes, then the resulting item is added to the crafting slot. The remaining 36 slots are generic inventory slots where the player can store items. We denote the crafting slot as [0], the 3×3 crafting grid as slots [A1] to [C3] where the letter denotes each row and the number denotes the column, and the remaining inventory slots as slots [I1] to [I36]. See Figure 2 for an example of a text description of the inventory that uses our annotation scheme.

Plancraft provides two types of observations:

- **text-only**: Agents receive a text description of the current inventory and crafting goal. The inventory is perfectly represented using our annotation scheme.

- **multi-modal**: Agents receive an image (high resolution with dimensions 664×704) that perfectly contains the crafting GUI. The goal is still provided as text.

3.3 Task Setup

The tasks within Plancraft all involve crafting specific items using a predefined set of available resources. The goal of each task is expressed in natural language, such as '*Craft an item of type: green_bed*'. The complexity of these tasks varies, ranging from single-step crafts (e.g., crafting wooden planks 🪵) to multi-step plans that require chaining multiple crafting recipes and actions (e.g., crafting a bed 🛏 first requires planks 🪵 and wool 🧶).

To generate crafting tasks, we represent the dependencies between items as a tree where each item is crafted from a set of required resources. The target item is the root of the tree, while the leaves and intermediate nodes represent the materials required to craft the item. For each crafting task, we sample the required recipes for the target item and then recursively explore the necessary materials to craft it. This builds a step-by-step plan from raw materials to the final product, simulating the crafting processes with varying levels of complexity. In addition to the necessary crafting materials, we also sample a number (4, 8, 16) of distractor items to make identifying the correct crafting path more challenging.

Finally, we validate that all generated tasks can be solved in the text-only and multi-modal environments using a planner (see Section 3.4) and exclude any invalid examples. We exclude any paths where the handcrafted planner's trajectory exceeds 30 steps.

Dataset Statistics

We define a complexity metric as directly proportional to the number of items used and the number of recipes needed to craft the item. We assign each example trajectory to a complexity bin, from least to most complex (very easy, easy, medium, hard, very hard), and ensure an equal proportion of examples from each bin distribution.

In total, we sample 1145 training examples, 570 validation examples, and 580 test examples. We hold out items such that 79 items in the validation set are not seen in the training set, and similarly 128 items in the test set do not appear in the training set, 63 of which also do not appear in the validation set. Since these evaluation sets are extensive, we also provide smaller subsets of the validation (110 examples) and test set (117 examples) which we use in our evaluation.

Environment Feedback

Since all models in Plancraft can handle text, we propagate formatting errors as text observations. This feedback informs agents about specific issues, allowing agents to adjust predicted outputs accordingly. For example, if an agent attempts to move an item from one slot to another but specifies the same source and destination slots, the environment will return a message indicating that these slots must be different. Or, if an agent generates a quantity that exceeds defined limits, the system provides an error message explaining that the quantity must fall within acceptable parameters. This feedback mechanism allows agents to correct their actions and ensures smoother interactions with the environment.

We implement an early stopping mechanism to accelerate evaluation – if an agent fails to craft any new item after $s = 10$ environment steps, then it is considered stuck. We also stop the episode after a maximum number of steps $n = 80$ is reached (recall that all examples can be solved in < 30 steps).

Impossible Tasks

Since we also wish to evaluate whether agents can predict when a task is unsolvable, we include a portion (20%) of the dataset that requires the agent to craft an item, which is impossible given the inventory. To generate this dataset, during sampling, key materials from the path are deliberately removed from the inventory. This is designed to test an agent's ability to recognise when a task cannot be completed given the available resources. As mentioned, all previous interactive environments (see Table 1) rely on the assumption that the task given to an agent is solvable. However, real-world scenarios are unlikely to meet this assumption and tasks may often be unsolvable due to missing resources or constraints.

To evaluate the agent's ability on impossible tasks, we introduce the impossible action, which when emitted by an agent will stop the episode. An impossible task is considered successful if and only when the agent emits impossible. For the impossible set of tasks, we report the F1 score of emitting impossible since it is now also possible for the agent to interrupt a solvable task (see Section 4.1).

3.4 Expert Planner

To establish a baseline, we handcraft a planner to find an efficient sequence of actions required to complete each task. This
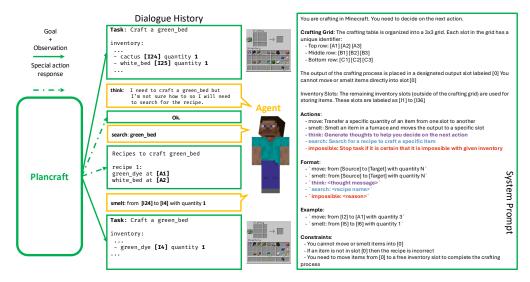
Figure 3: Example flow of Plancraft. The agent can use the varying set of tools as described in its system prompt. The action descriptions and formats in the system prompt are only present if the action is allowed. The dialogue history is passed as a sequence of messages. Environment messages include the task goal and current observation (text or image). If the agent takes an action that does not impact the environment such as `think` or `search`, then the environment does not return an observation but a response based on this action.

planner serves as a standard against which the performance of various agent models can be benchmarked.

We implement the solver as a memoized Depth-First search over all possible crafting recipes given an inventory. If found, we return the shortest path between the initial inventory and the target object that needs to be crafted as a series of `smelt` and `move` actions. If the solver does not find a shortest path after 30 seconds, we time out the search. Note that for simplicity our planner does not `smelt` multiple items in a single action and does not move items directly into the crafting grid from the crafting slot [0] or from a `smelt` action. As a result, it is possible for shorter paths to exist if an agent is able to group multiple smelting actions into one or anticipate the location of where an item needs to be placed after crafting/smelting. The planner allows us to evaluate the length of a generated plan against a handcrafted standard. We also use the solver to set hyper-parameters, such as the maximum number of steps an agent can take to complete a task.

3.5 Integrating External Knowledge

Similarly to Fan *et al.* [2022], Plancraft includes the Minecraft Wiki as a natural language knowledge base, allowing agents to perform RAG to assist in crafting decisions. This addition enables the agents to use external knowledge, testing their ability to retrieve relevant and multi-modal information and apply it to the planning problem. We scrape the pages from the Minecraft Wiki that contain recipes and post-process the data to convert them to Markdown format.

We implement a `search` action that allows an LLM to search for a given item in the knowledge base. Since we wish to evaluate the limits of RAG, we design the `search` operator to return a gold-label instantiation of a recipe required to craft the particular item. Therefore, our RAG mechanism helps estimate the performance of our system under a perfect retriever and parser.

4 Method

We evaluate both the open-source Llama 3.1 8B and Llama 3.3 70B models [Dubey *et al.*, 2024] as well as gpt-4o-mini[2] [OpenAI, 2024] in our Plancraft environment. All tested models are chat models, and we format the environment observations and feedback as messages from a user role. We show the interactions between the agent and the Plancraft environment in Figure 3. We evaluate each model in the text-only and multi-modal environments, test making available different sets of actions or tools, and measure the impact of zero-shot versus few-shot versus fine-tuning. We implement the ReAct [Yao *et al.*, 2023b] 'thought' strategy as a tool, where the agent can emit `think` as an action within the environment. Any generated text after `think` is ignored.

Tool use We evaluate multiple sets of different tools outside of the core environment actions (`move` and `smelt`), we test whether including `think`, an oracle recipe `search` (RAG) and `impossible` has an impact on the planning capabilities of agents. If a tool is active, we incorporate a description of it within the system prompt and provide an example of its use in the few-shot examples (see Figure 3).

Fine-Tuning We fine-tune a Llama 3.1 8B model on oracle planning trajectories from the training set using LoRA [Hu *et al.*, 2021] ($r = 64, \alpha = 32$). These trajectories only consist of Observation, Action pairs directly obtained from our handcrafted planner and only include the `smelt` and `move` actions. As a result, we also test whether the fine-tuned LLM can take advantage of new actions that it has not been explicitly trained on.

Image Observations We evaluate models on both text and image observations. Since the Llama models we evaluate are

---

[2]gpt-4o-mini-2024-07-18

text-only models, we train a custom pre-trained bounding-box model (Faster R-CNN) to map from images to a list of items, quantities, and positions. We train the bounding-box model separately, sampling random inventories and their accompanying bounding boxes to obtain training data. Using the features of the bounding box, the model is trained to classify not only an object's class, but also its quantity.

We then map the extracted symbolic representations to the same format as our text-only observations. We use this approach to benchmark how much planning ability is lost if the observation is generated from an imperfect classifier. For gpt-4o-mini, we evaluate using these extracted textual observations from the bounding-box model and also test passing image inputs directly (since gpt-4o-mini supports images).

4.1 Metrics

For all methods, we report the task success rate – whether or not the goal item has been crafted, the plan length – the number of *environment* actions the agent has taken (i.e. smelt and move), and the total number of tokens used (both input and output combined). Task success evaluates the percentage of tasks completed successfully by the agent.

Using the expert planner, we can compare an agent's trajectories compared to those of the expert. We calculate the average Action Efficiency (AE) as the ratio of the number of actions in an agent's plan P compared to the path of the expert planner P^e. Note we only consider successful plans in the calculation of this metric:

$$\text{Action Efficiency} = \frac{\sum_{i=1}^{N_{\text{successful}}} (P_i - P_i^e)}{N_{\text{successful}}} \qquad (1)$$

We measure whether knowledge, internal and external, is used through the number of think and search calls. When we evaluate the performance of models when the impossible is allowed, we also report the F1 score of emitting the action. Since predicting impossible immediately stops the episode, the impossible setup introduces a new form of failure mode where a model can emit an impossible when faced with a solvable problem.

Lastly, we track the overall compute efficiency using the total number of tokens used as a proxy.[3] Token usage is heavily impacted by tool use since calling external tools and actions both requires the model to generate additional tokens (to call the tool) but also introduces new information as a result of the external call. However, we note that tools could hide additional compute costs not factored into simple token usage.

Unless specified, all models are evaluated under the same conditions, with a fixed number of trials per task and the same prompts. We use a temperature of $t = 0.6$ and run five generations per model to report an average on each metric.

5 Results

We report results with text-only observations in Table 2 and multi-modal observations in Table 3.

---

[3]We consider input and output tokens as equivalent.

5.1 Act and ReAct baselines

We first restrict the action space to only the environment actions, move (M) and smelt (S), and evaluate the planning capabilities of Llama 3.1 8B, Llama 3.3 70B and gpt-4o-mini in a few-shot setting. We find that gpt-4o-mini and Llama 70B perform similarly, with an overall task success rate of 0.11 and 0.16 respectively. Llama 8B performs noticeably worse with a success rate of only 0.04. In terms of plan efficiency, we find that gpt-4o-mini is closer to the expert planner (0.29) than the Llama models (0.86 and 1.09), which indicates that when the model succeeds, it does so efficiently.

We then extend the pool of possible actions to include think, similar to a ReAct strategy [Yao *et al.*, 2023b]. Note that we also update the few-shot prompt appropriately to include a description of the action and an example of its use (see Figure 3). We restrict the maximum number of consecutive think actions to three; however, we find that, in practice, the model does not get stuck in a continuous thought loop. Using think, the few-shot models slightly improve in overall task success: Llama 8B 0.06(+0.02), Llama 70B 0.22(+0.06), gpt-4o-mini 0.12(+0.01). However, think also comes with added costs, as the model is now allowed unconstrained generation steps that lead to longer dialogues. We can see the direct effects of this on token usage; for instance, gpt-4o-mini increases its token usage from 8.2M to 11.6M tokens.

5.2 External Knowledge

When we allow search, we enable the models to incorporate the external knowledge contained in the Minecraft Wiki through the RAG oracle retriever. We find that this has substantial benefits for all three few-shot models, boosting Llama 7B to 0.30(+0.24), Llama 70B to 0.53(+0.31) and gpt-4o-mini to 0.27(+0.15) in overall success rate. As with think, search introduces additional steps and therefore we would expect a rise in average tokens used, however, we find that is not the case. Lower token usage can be partially explained by a global decrease in the use of think when search is available. Instead of speculating on how to craft a particular recipe, search allows models to directly retrieve precise information. This reduces the need for additional reasoning steps, resulting in more efficient token usage.

5.3 Predicting Impossible Tasks

When we include impossible as a possibility, we introduce a way for the agent to interrupt an episode. We find that this negatively affects the smaller Llama 8B model (-0.12) more than the larger Llama 70B ($+0.02$) and gpt-4o-mini ($+0.01$) in overall success rate. We also record the F1 rate of emitting impossible, and find that the best performing model, the model that can most accurately predict when a task is impossible or not, is the Llama 70B model, with an F1 score of 0.64. Adding impossible also decreases the usage of tokens, as models can now interrupt an episode when stuck. This reduces the average tokens used by more than half for some models; for example, gpt-4o-mini goes from 10.9M to 4.7M in the average tokens used.

Tools	Model	Success Rate (↑)				Avg. Action Count			Impossible F1 (↑)	Avg. Plan Length (↓)	AE (↓)	Avg. Tokens Used (↓)
		Difficulty			Overall							
		Easy	Medium	Hard		think	search	impossible				
M S	Llama 8B	0.10	0.00	0.00	0.04	-	-	-	-	13.21	0.86	6.2M
M S	Llama 70B	0.37	0.04	0.00	0.16	-	-	-	-	15.94	1.09	8.9M
M S	gpt-4o-mini	0.26	0.00	0.00	0.11	-	-	-	-	14.98	0.29	8.2M
M S	Llama 8B FT	<u>0.67</u>	<u>0.45</u>	<u>0.12</u>	0.41	-	-	-	-	10.14	0.03	4.1M
M S T	Llama 8B	0.15	0.00	0.00	0.06	9.50	-	-	-	21.80	3.87	11.1M
M S T	Llama 70B	0.44	0.15	0.03	0.22	10.57	-	-	-	21.16	2.50	11.8M
M S T	gpt-4o-mini	0.29	0.00	0.00	0.12	13.98	-	-	-	23.53	2.37	11.6M
M S T	Llama 8B FT	<u>0.57</u>	<u>0.42</u>	<u>0.12</u>	0.37	0.06	-	-	-	10.41	0.07	4.5M
M S T SE	Llama 8B	0.57	0.31	0.01	0.30	4.75	3.18	-	-	18.41	3.38	9.3M
M S T SE	Llama 70B	<u>0.79</u>	<u>0.60</u>	<u>0.23</u>	0.53	6.69	2.28	-	-	18.71	4.27	9.8M
M S T SE	gpt-4o-mini	0.56	0.21	0.00	0.27	11.16	2.15	-	-	21.98	4.91	10.9M
M S T SE	Llama 8B FT	0.61	0.39	0.12	0.38	0.00	0.01	-	-	10.36	0.11	4.8M
M S T SE I	Llama 8B	0.40	0.09	0.01	0.18	2.31	2.36	0.72	0.32	9.77	2.53	4.9M
M S T SE I	Llama 70B	<u>0.80</u>	<u>0.65</u>	<u>0.23</u>	**0.55**	5.20	2.20	0.16	0.64	16.76	4.05	9.6M
M S T SE I	gpt-4o-mini	0.56	0.23	0.00	0.28	6.45	1.15	0.48	0.42	14.89	4.94	4.7M
M S T SE I	Llama 8B FT	0.58	0.40	0.11	0.36	0.00	0.02	0.00	0.00	10.38	0.09	5.7M

Table 2: Average Success Rates (SR) for different models and sets of actions available: move (M), smelt (S), think (T), search (SE), impossible (I). We also report the average count of special actions used, the average plan length, the Action Efficiency (AE) and the average total number of tokens used. We group very easy and easy difficulty categories together as 'easy' and hard and very hard as 'hard'.

Model (M S)	Overall Success Rate (↑)	Avg. Plan Length (↓)	AE (↓)	Avg. Tokens Used (↓)
Llama 8B R-CNN	0.00	13.35	3.00	5.3M
Llama 70B R-CNN	0.06	14.47	1.00	6.1M
gpt-4o-mini R-CNN	0.03	14.93	0.23	5.8M
gpt-4o-mini IMG	0.01	15.53	2.60	95.3M

Table 3: Results for different models with the basic sets of actions (M S) and image observations. We use our R-CNN to extract the symbolic representation from images. Since gpt-4o-mini also supports image inputs, we evaluate the model as a unified VLM where observations are passed directly as images (gpt-4o-mini IMG).

5.4 Effects of Fine-tuning

As mentioned in Section 4, we also fine-tune an LLM on expert plans (Llama 8B FT). As in Wang *et al.* [2022a], we find that smaller models can increase their task success with only about 1k training examples. Llama 8B FT obtains an overall success rate of around 0.41, which decreases slightly with each additional action we introduce. We find that fine-tuning severely decreases the model's ability to use new actions. Llama 8B FT almost never uses think, search, and impossible, and thus is also our most efficient model in terms of plan length.

5.5 Image Observations

In Table 3, we use images instead of text inputs. We leverage the pre-trained bounding-box Faster R-CNN model to map images into text descriptions, and confirm a drop in accuracy for all models: Llama 8B (-0.04), Llama 70B (-0.10), and gpt-4o-mini (-0.08). Because gpt-4o-mini supports image inputs, we also evaluate passing images directly to the model instead of the text observations. In this case, the system instructions remain the same, but we modify the few-shot prompt to replace the observations with images. We refer to this setup as gpt-4o-mini IMG and note that it is unable to solve our task (0.01) and uses 100 times as many tokens, since every image is converted into a long sequence of tokens.

6 Conclusion

We introduce Plancraft, a multi-modal dataset to evaluate the planning capabilities of LLM-based agents. Using Minecraft crafting, Plancraft allows for the assessment of both task accuracy and an agent's ability to judge task feasibility. By including solvable and intentionally unsolvable tasks, the dataset provides a controlled setting for examining agentic behaviour in step-based problem-solving contexts.

Our results confirm that larger models, such as Llama 70B, outperform smaller models like Llama 8B in both task success and efficiency in Plancraft. This advantage is further amplified by specialised actions such as think and search. The introduction of impossible highlights a trade-off: while enabling models to recognise unsolvable tasks reduces token usage and increases efficiency, it disproportionately impacts smaller models like Llama 8B. Fine-tuning smaller models, as shown with Llama 8B FT, significantly boosts task success but also reveals a key limitation: the inability to effectively use new actions like think and search, indicating fine-tuning may overly constrain behaviour. Additionally, our findings emphasise challenges of multi-modal settings; while text-only tasks highlight strong performance trends, raw image inputs, as in gpt-4o-mini IMG, underscore fundamental limitations of out-of-the-box VLMs for effective planning. Overall, our results underscore the complexity of bridging multi-modal inputs and decision-making in planning tasks.

Acknowledgments

References

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. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24639–24654. Curran Associates, Inc., 2022.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

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

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. (arXiv:2207.05608), July 2022. arXiv:2207.05608 [cs].

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, page 9459–9474. Curran Associates, Inc., 2020.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

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

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. (arXiv:2112.09332), June 2022. arXiv:2112.09332 [cs].

OpenAI. Gpt-4 technical report. (arXiv:2303.08774), March 2024. arXiv:2303.08774 [cs].

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. (arXiv:2205.12255), May 2022. arXiv:2205.12255 [cs].

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. (arXiv:2305.15334), May 2023. arXiv:2305.15334 [cs].

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. *arXiv preprint arXiv:2311.05772*, 2023.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. November 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. November 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALF-World: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. September 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*, 2023.

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

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Agentgym: Evolving large language model-based agents across diverse environments, 2024.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. (arXiv:2305.18752), May 2023. arXiv:2305.18752 [cs].

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc., 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, page 11809–11822. Curran Associates, Inc., 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.