Blocks, Bots, and Bottlenecks: Studying Real-time and Adaptive Multi-Agent LLM Collaboration

Isadora White[1*], **Kolby Nottingham**[2*], **Ayush Maniar**[1], **Max Robinson**[3], **Hansen Lillemark**[1], **Mehul Maheshwari**[1], **Lianhui Qin**[1], **Prithviraj Ammanabrolu**[1]
∗ equal contribution
[1]University of California, San Diego
[2]Latitude Games [3]Emergent Garden

Abstract

Collaboration lies at the heart of human intelligence—whether brainstorming ideas, dividing responsibilities, or planning complex tasks together. Can large language models (LLMs) do the same? We introduce MINDcraft, a dynamic platform that pushes the limits of AI collaboration by combining real-time, adaptive communication with 47 powerful in-game tools that let agents act in the rich, open world of Minecraft. Alongside it, we present MINECOLLAB, a benchmark for evaluating how well agents coordinate, plan, and execute tasks together. Our experiments reveal a striking result: LLM agents falter when collaboration demands clear and detailed communication—showing up to a 15% performance drop when they must articulate step-by-step plans. These findings highlight that while today's agents can act, true collaboration still hinges on mastering language as a medium for shared understanding and joint reasoning. Video demonstrations illustrating the capabilities and failure modes of our agents can be found here: `https://mindcraft-minecollab.github.io/index.html`

1 Introduction

Imagine AI agents that can communicate in real-time, adapt to their roles and collaborators dynamically, while coordinating with each other on complex multi-step tasks in rich 3D environments. Despite LLM-based agents winning math OLYMPIAD gold [1], relatively little progress has been made towards this vision. AI Agents still struggle with the essential challenges of (1) collaborating with other agents [2] (2) long horizon planning [3] and (3) manipulation of an embodied environment [4]. This raises a provocative question: can AI agents learn to collaborate as fluidly as humans, or are there fundamental barriers in our current approaches?

Current multi-agent tasks and frameworks often focus on a hierarchical structure of agents (e.g. with leader and follower agents) and a strict turn-based communication structure - preventing agents from executing multiple actions in a row, interrupting their actions, or communicating while executing actions. Moreover, many existing platforms do not focus on the combination of collaborative reasoning, long horizon planning, and manipulation of an embodied environment through complex tool calls, as is the subject of this paper. While we believe that more localized studies are valuable, realistic communication often occurs in real-time and adaptively over long horizons and in complex environments. For example, when two researchers co-author a paper, they exchange messages asynchronously, interrupt each other, and make collective decisions regardless of hierarchy. In fact, when humans collaborate they frequently interrupt each other and take multiple turns at a time. [5]

[*]These authors contributed equally to this work.

Preprint. Under review.

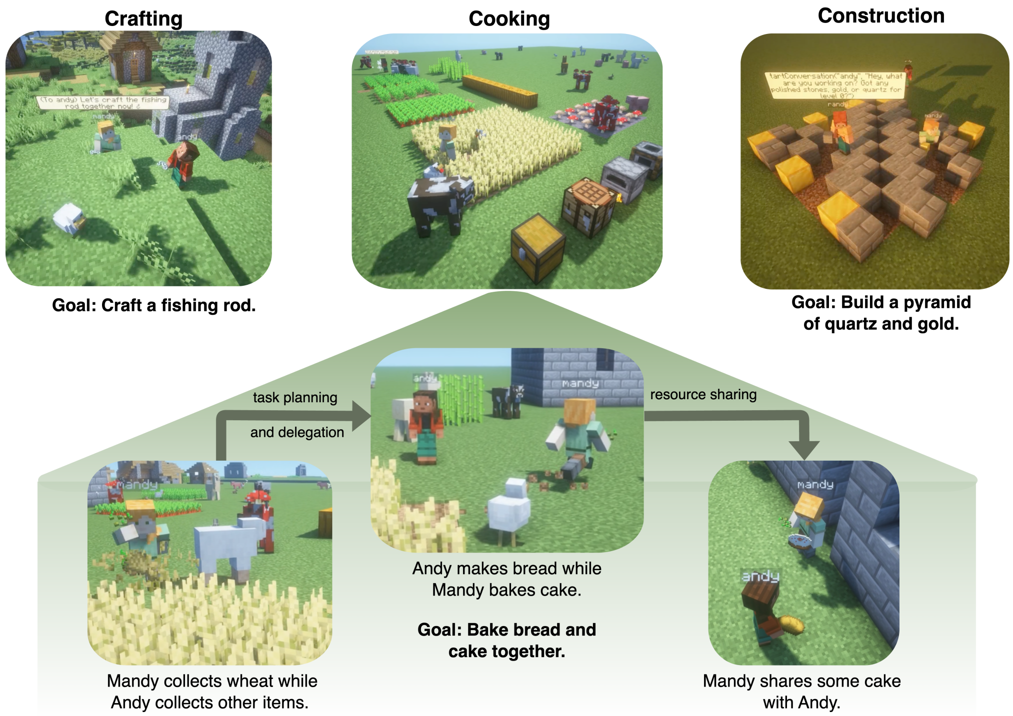


Figure 1: **Task suites and challenges.** In this figure, we see the collaborative and embodied reasoning challenges displayed. In the cooking and crafting tasks, the agents need to delegate tasks, share resources and use embodied planning to manipulate the world of Minecraft. In the construction tasks, the agents need to navigate and coordinate in the space to ensure they consistently build towards their objective without undoing any progress the other agents have made. All together these tasks comprehensively test collaborative and embodied reasoning.

To fill this gap, we introduce MINDcraft, an agentic framework in Minecraft that supports real-time, hierarchy-free, adaptive communication. Further, we build 47 custom tools for MINDcraft, the most extensive API for controlling Minecraft to our knowledge. These tools enable us to fully unlock Minecraft's potential as a complex, open-ended testbed for embodied AI collaboration. To enable increased agent performance over longer horizons, we introduce a summarization-based memory structure and RAG over relevant few-shot examples. To quantify the effectiveness of our agents, we unveil the MINECOLLABsuite of tasks that tests the combined challenges of long horizon collaborative reasoning in an embodied environment with real-time adaptive communication. In designing MINECOLLABwe think strategically about designing tasks where collaboration and communication are *critical* for task success. We introduce three tasks: cooking, which involves preparing a meal while coordinating ingredient collection; crafting, where agents assemble tools (for example a bookshelf) from available materials; and construction, where agents must build procedurally generated blueprints from blueprints as can be seen in Figure 1.

We systematically evaluate LLM agents on these three task suites, testing their communication and collaboration capabilities in a complex environment. First, we demonstrate that for our MINECOLLABtasks, communication and collaboration is critical to task success - with performance dropping dramatically with only one agent or without communication. However, we find that task success is highly sensitive to communication quality, with performance dropping by 15% when agents must explicitly communicate detailed plans. On our collaborative task suite, human collaboration remains more flexible and efficient than collaboration demonstrated by LLM agents significantly outperforming AI agents.

Our contributions are threefold: (1) MINDcraft — the first framework for adaptive, real-time collaboration in Minecraft, (2) MINECOLLAB— a benchmark suite that evaluates collaborative reasoning and adaptive real-time communication, (3) a comprehensive evaluation of LLM agents on MINECOLLAB,

Table 1: **Comparison to Other Multi-Agent Platforms**, we illustrate the difference between our benchmark and other popular platforms for studying multi-agent coordination or embodied agents. **Long horizon** refers to the complex sequence of actions (on average over 20 steps) that need to be taken in order to accomplish our task objectives. **Embodied** refers to the ability of AI agents to manipulate the physical world - or a complex simulation thereof. **Flexible Roles** refers to a peer-to-peer communication structure where agents can take on any role regardless of the situation. **Real-time** refers to the ability of the agents to take actions asynchronously and simultaneously, not relying on a fixed turn-based structure.

Platform	Human-AI	Long Horizon	Embodied	Flexible Roles	Tools	Real-time
Overcooked [6]			✓	✓		✓
LLM-Coord [7]	✓					
PARTNR [8]			✓	✓		
Habitat AI [9]	✓		✓	✓		
CerealBar [10]	✓	✓		✓	✓	
Generative Agents [11]		✓	✓		✓	
MineLand [12]			✓	✓		
CoELA [13]	✓		✓	✓	✓	
Odyssey [14]		✓	✓	✓	✓	✓
MINDcraft (ours)	✓	✓	✓	✓	✓	✓

revealing the benefits of flexible communication and tool calling, and highlighting the bottlenecks in collaboration and communication.

2 Related Work

Minecraft as a Tool for AI Research. Minecraft is a vast open-ended embodied world with complex dynamics and sparse rewards. For these reasons, it has been a popular tool for researchers for studying world models [15], planning [16, 17, 18, 19], theory of mind and structured collaboration [20, 12, 21, 22, 23, 24, 25, 26] or in adversarial tower defense scenarios [27]. We chose Minecraft for similar reasons, the expressivity of the simulator allows for a large range of tasks to be designed.

Platforms for Multi-Agent and Human-AI Collaboration. Overcooked AI [6] is a popular framework for studying the capabilities of AI agents to collaborate with one another. Similarly, CerealBar [10] and GovSim [28] test collaborative abilities of LLM agents but through a different lens. Other works such as [29, 30, 31], study fine-grained collaboration between people. We use these simulators as inspiration and build on them in the following ways: 1) in addition to cooking themed tasks, we include a greater variety of tasks such as crafting and construction tasks, 2) create an environment that is controllable in language, 3) study both the peer-to-peer and leader/follower interactions of our agents. Table 1 outlines the differences between our platform and others.

Embodied AI and Robotics for Collaboration. Embodied scenarios in indoor home environments are the most common type of task previously studied [4, 32, 9, 33]. Habitat AI and the corresponding dataset of instructions PARTNR [8] creates a large dataset of human-AI collaborative tasks. Similarly, [34] studies how agents can collaborate in a kitchen.

Aligning Text and Visual Worlds: ALFWorld [35] is another simulator that has dual modes for perceiving the world in a text-only world in an embodied visual environment. Works such as ALFWorld have been used to improve performance of vision-based agents by aligning them better with the text-based policy [36] . MINDcraft's simultaneous support for vision and textual inputs could similarly allow for improved embodied agents by distilling from a text-based policy

Multi-agent Methods for LLMs. Frameworks like Teach [37], Optima [38] and others [39, 40] add dimensions of parallelized execution and optimized efficiency in multi-agent systems. In [34] and [12], the authors focus on creating a modular and iterative prompting method. Alternatively, [41] and [42] propose new finetuning approaches for improving multi-agent interaction in language models. One objective of MINDcraft is to promote research into these methods by providing complex embodied environments to study multi-agent communication.

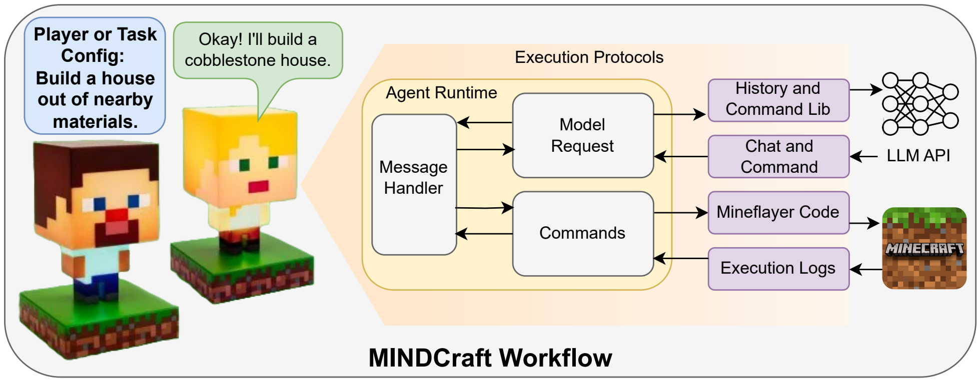


Figure 2: Overview of the MINDcraft workflow. A user or task configuration (left) provides instructions (e.g., "Build a house out of nearby materials"). The *Agent* (center) takes these instructions, consults an LLM (via a model request) and invokes high-level commands/tools. These commands are then executed in the Minecraft environment (right), with the agent receiving feedback through execution logs. The extensive command library in MINDcraft enables flexible, plug-and-play experimentation with collaborative and embodied LLM agents in a partially observable Minecraft world.

3 MINDcraft

MINDcraft is a robust and adaptable platform designed for running experiments in the grounded environment of Minecraft. The platform introduces 47 custom-made tools that unlock the full power of Minecraft as a virtual world for agents. The full list of tools can be found in Section 15. Moreover, MINDcraft's communication structure is peer-to-peer and asynchronous in contrast with much of the previous literature that focuses on hierarchical structures and synchronous communication. This allows agents in MINDcraft to develop roles interactively and adaptively and asynchronously execute actions while in conversation with other agents.

3.1 State and Action Spaces

State Space. The prompt given to MINDcraft agents can be found in section 10. This prompt includes information about the agents current location, inventory, and health stats as well as the surrounding blocks. To get access to visual inputs, the agent can call the `!look` tool to get a screenshot of the current state. We show in an empirical study that the inclusion or exclusion of the look tool does not change performance much for gpt-4o-mini on 2 agent tasks in Section 17.

Action Space(s). There are several previously used levels of abstraction for action spaces in Minecraft. The lowest level of actions such as the MineRL competition action space [43], which AI research in Minecraft has historically depended on, are actions such as "jump," "look up," or "use". While such actions are closer to how a human player interacts with Minecraft through mouse-and-keyboard inputs, these low-level interactions required extensive bespoke training of AI agents through RL and computer vision architectures [44]. The Mineflayer API [45] introduces a set of higher-level commands and abstractions in Javascript, such as the `pathFinder` module, which enables a bot to navigate from its current location to another player or specific coordinates (x, y, z). This API allows AI systems to interact with Minecraft using high-level code, offering a more abstract and programmatic approach to controlling gameplay and bot behavior but does not reflect how the average human player interacts with this environment [16].

Our contribution, MINDcraft enhances these abilities further and bridges the gap between human-like actions and ease of programmatic AI interactions with the Minecraft environment by building a set of 47 parameterized tools that can be directly invoked by LLMs. For example, instead of generating Mineflayer code to 1) find the nearest player named "randy," 2) travel to its location, 3) identify oak_logs in the bot's inventory, and 4) drop four oak_logs for randy to pick up, the LLM can simply output `!givePlayer("randy", "oak log", 4)`. This abstraction empowers LLMs to reason over a higher-level sequential action space, enhancing their ability to perform complex tasks within

Minecraft. When necessary, MINDcraft supports a tool that permits the LLM agent to output custom Mineflayer code in Javascript to perform custom actions or build buildings. The list of 47 high level actions we designed can be found in Appendix 15.

3.2 Agent Architecture

The MINDcraft architecture includes 4 main components (1) a library of 47 high-level action commands and observation queries, (2) an asynchronous communication structure for multi-agent dialogue (3) a server for launching and managing agents, (4) the main agent loop for handling messages from players and other agents, (5) a layer for prompting and calling arbitrary language models with support for 13 different APIs including vllm, ollama, openAI, anthropic among others (6) a module for guiding custom code generation and (7) guided self-prompting for the agents for self-guided play.

To further enhance the abilities of our agents, we enable the ability of our agents to retrieve and prompt with few-shot examples showing usage of our tools via embedding similarity to the current conversation—essential for enhancing the abilities of LLMs via an embodied Retrieval Augmented Generation (RAG) system [46]. To prevent the context of the model from growing too long during the course of our very long trajectories lasting over 75 messages on average as shown in Table 2, we introduce a memory framework. Once the length of the message stream - including messages from other bots and system responses to code reaches a length of over 15, we prompt the LLM to summarize the first five utterances and the previous memory into a 500 word summary. When an agent wishes to execute an action that exceeds the parameters of our existing tools (e.g. for building large free-form buildings), they can use the `!newAction` command which comes with it's own library of few-shot examples that we use retrieval for as well. The prompts for coding, memory, as well as the few-shot examples can be found in section 10. The robust agent architecture of MINDcraft allows us to evaluate LLMs of varying quality while focusing on our collaborative benchmark MineCollab.

3.3 Multi-agent Collaboration

Unlike previous multi-agent systems that rely on highly structured, synchronous communication protocols with centralized control (cite), our framework adopts a peer-to-peer, asynchronous communication model. We argue this design is essential for realistic collaboration in dynamic environments such as MINDcraft, where agents must interleave planning, action execution, and dialogue in real time.

In our system, conversations are managed by a lightweight conversation manager. Agents can initiate or end dialogues at will using the `!startConversation` and `!endConversation` commands. At any point, an agent receiving a message may respond immediately, delay a response, ignore it, or initiate a separate conversation with another agent. This flexibility enables communication to overlap naturally with ongoing actions rather than forcing agents into rigid turn-taking.

Although only two agents can be engaged in an active conversation at once, our pairwise protocol scales seamlessly to multi-agent settings by allowing agents to transition between conversations as tasks evolve. Importantly, action and communication proceed concurrently: while one agent executes an action, the other may send a message, which is delivered once both agents are able to process it. If both are acting simultaneously (e.g., placing blocks), the conversation is paused until actions complete, after which dialogue resumes.

This asynchronous design provides two key advantages. First, it avoids bottlenecks that would arise under synchronous communication, where ongoing actions would stall progress for all agents. Second, it mirrors natural human collaboration, where partners communicate opportunistically while continuing with their work. Rather than enforcing rigid interaction protocols, our framework supports open-ended, real-time coordination, which we view as a critical step toward scalable and ecologically valid studies of LLM-based collaboration.

4 MineCollab - Collaborative Embodied Task Suite

We introduce MineCollab, an example of a benchmark that can be built in MINDcraft. The MineCollab benchmark involves three practical domains specially designed to require collaboration: cooking, which involves preparing a meal while coordinating ingredient collection; crafting, where agents

Table 2: **Summary of our train and test tasks sets. Train** and **test** denotes the number of unique train and test tasks used for generating data and evaluating performance. **Trials** denotes the number of times we ran the train tasks using our oracle agent (llama3.3-70b-instruct). **Success** denotes the number of these trials that were successful, **examples** is the number of transitions in those successful trials, and **Avg. Traj. Len** is the average number of steps per episode across successful and unsuccessful collected tasks. Since the dataset and tasks are procedurally generated, we can generate more data by sampling using a more powerful model and then filtering based on success.

Task	Train	Test	Trials	Success	Examples	Avg Traj. Len.
Cooking	350	54	1942	349	59444	74.5
Crafting	1,200	100	2176	366	19229	39.4
Construction	2,000	30	211	52	9228	111.5
Total	3,550	184	4329	767	87901	75.1

assemble furniture and tools from mined materials; and construction, which requires building structures from detailed blueprints. These domains reflect real-world scenarios and pose substantial challenges, requiring agents to execute long-horizon action sequences (on average over 20 steps), interact effectively with their environment, and communicate and coordinate with other agents under resource and time constraints. We test agents on multiple individual tasks within each domain that are procedurally generated along a carefully crafted set of dimensions designed to elicit and evaluate embodied reasoning and collaboration abilities.

Cooking Tasks. At the beginning of a cooking task episode, the agents are initialized with a goal to make a meal, e.g. they need to make cake and bread. The agents then need to coordinate the collection of ingredients through natural language communication (e.g. Andy collects wheat for the bread while Jill makes the cake) and combine them in a multi-step plan. To assist them in collecting resources, agents are placed in a "cooking world" that includes everything from livestock, to crops, to a smoker, furnace, and crafting table. Additional items that are not present in the "cooking world" but are necessary for the task at hand are split between their inventories to ensure collaboration necessity. For example, if they need to make cake one agent will have access to an egg, the other three milk buckets. Following a popular test of collaboration in humans, we further introduce a “Hell’s Kitchen” variant of the cooking tasks where each agent is given the recipes for a small subset of the items they need to cook and must communicate the instructions with the other teammates. For example, if the task is to make a baked potato and a cake, one agent is given recipe for baked potato, but is required to bake the cake to complete the task, forcing them to ask their teammate for help in baking the cake. Agents are evaluated on whether are successfully able to complete the set requirements to make the recipes. The environment and objectives of the tasks are randomized every episode.

Crafting Tasks. Crafting has long been the subject of Minecraft agent research [43]—our crafting tasks encompass the entire breadth of items that are craftable in Minecraft including clothing, furniture, and tools. At the beginning of each episode, the agents are initialized with a goal (e.g. make a bookshelf), different sets of resources (e.g. books and planks), and access to a crafting recipe, that is occasionally blocked. To complete the task, the agents must: (1) communicate with each other what items are in their inventory; (2) share with each other the crafting recipe if necessary; and (3) give each other resources to successfully craft the item. To make the crafting tasks more challenging, agents are given longer crafting objectives (e.g. crafting a crossbow, which requires agents to first make a tripwire hook, and sticks then combine with string and iron to make the crossbow). Once again, each of these components can be controlled to procedurally generate tasks.

Construction Tasks In the construction tasks, agents are directed to build structures from procedurally generated blueprints. Blueprints can also be downloaded from the internet and read into our blueprint format - enabling agents to build anything from pyramids to the Eiffel Tower. We choose evaluate primarily on our generated blueprints as they provide fine-grained control over task complexity, allowing us to systematically vary the depth of collaboration required—e.g. number of rooms in the interior of palace, or the amount and types of materials required for each room. At the beginning of each episode, agents are initialized with the blueprint, materials (e.g. stone, wood, doors, carpets) in such a way that no agent has the full resources or the expertise in terms of the types of tools that can be used to process the resources and complete the entire blueprint. For example, if the blueprint required a stone base and a wooden roof, one agent would be given access and the ability to manipulate stone, the other to wood. Agents are evaluated via an edit distance based metric that judges how close their constructed building is to the blueprint and the metric reported in **??** is the average of those edit distance scores.

Train and Test Splits. To ensure experimental reproducibility, for each of our domains, we create and split the possible tasks into train and test tasks, taking special care to ensure that each subset are significantly different to avoid dataset pollution. For our construction tasks, we procedurally generate the blueprints for train and test tasks with different seeds, ensuring no two blueprints are identical. For the cooking and crafting tasks, we ensure that the train and test tasks involve different recipes and procuring different ingredients. This ensures that the same plan for making an item such as baking a cake is not present in both splits. Item division for cooking tasks can be found in the Appendix 14.2.

5 Methods

SFT Dataset Creation. We also provide users with tools to generate behavior cloning (or Supervised Fine Tuning, SFT) data that can be used to train LLMs further. The generating oracle agent is run on the train tasks, and then the data is filtered based on whether the run has been successful. Then, we use each transition in the trajectory as a data point. We chose to generate data from llama3.3-70b-instruct, because of its reasonable performance on the benchmarks (Figure 3 and its open-weight nature ensuring a higher standard of reproducibility for our benchmark)—but note that such an oracle agent can be any other LLM or even a human player. For crafting and cooking tasks where final scores are binary 1 or 0, we only take successful runs, and for construction tasks where there is a continuous edit-distance based score, we take trials that score within the top 25% of all runs. Dataset examples can be found in Appendix 12 and statistics can be found in Table 2. Training llama3-8b-instruct on each of these task-specific datasets for one epoch with a learning rate of $1e-05$ improves performance over the base model by over 5% on the cooking task, matches performance on the construction task with the 70b model, and outperforms gpt-4o and llama3.3-70b-instruct on the crafting tasks by over 5%. By increasing the performance of less compute intensive models we hope to improve the accessibility of our benchmark and further research into small open source models.

Human Studies Using a group of 17 non-expert humans and 4 human co-authors with expert knowledge in Minecraft, we study how well they can collaborate with each other and with our agents. We find that in all task categories that the humans working with other humans and humans working in tandem with 1 to 3 AI agents are more successful than the AI agents working with other AI agents, getting 100% success in crafting tasks and 75% in cooking tasks and outperform the top performing model by 20% on construction tasks. This illustrates the power of expert human collaboration in comparison to LLMs.

Compute Resources Required. Our SFT training occurred on a node of 8 80GB H100s, and inference for open weights model was run on a node of 8 40GB A100s. To run our tasks on a large open weights model such as DeepSeek R1, one would require a multi-node cluster of at least 4 nodes of 8 H100s. The closed source models cost between one to two dollars per agent per task on the cooking and crafting tasks. Because of the large number of tokens needed to describe the blueprints and generate code to build buildings, the construction tasks generally cost between 2 and 3 dollars per task per agent on the closed source models gpt-4o and claude-3-5-sonnet.

6 Experiments

We compare the performance of current state-of-the-art open and closed weights LLMs on MineCollab. Namely, gpt-4o, claude-3-5-sonnet, llama3.3-70b-instruct, llama3-8b-instruct [47] and an open weights reasoning model r1-distill-llama [48]. Our study design and analysis rely on the modular design of MineCollab to vary task complexity along two dimensions: embodied reasoning, and collaborative communication.

How does human–AI collaboration compare to AI–AI performance on our tasks? As shown in fig. 3, human collaborators substantially outperform AI–AI pairs across all task categories. Both expert and non-expert participants achieve markedly higher success rates when paired with an AI partner or another human, demonstrating that humans can effectively guide and compensate for model

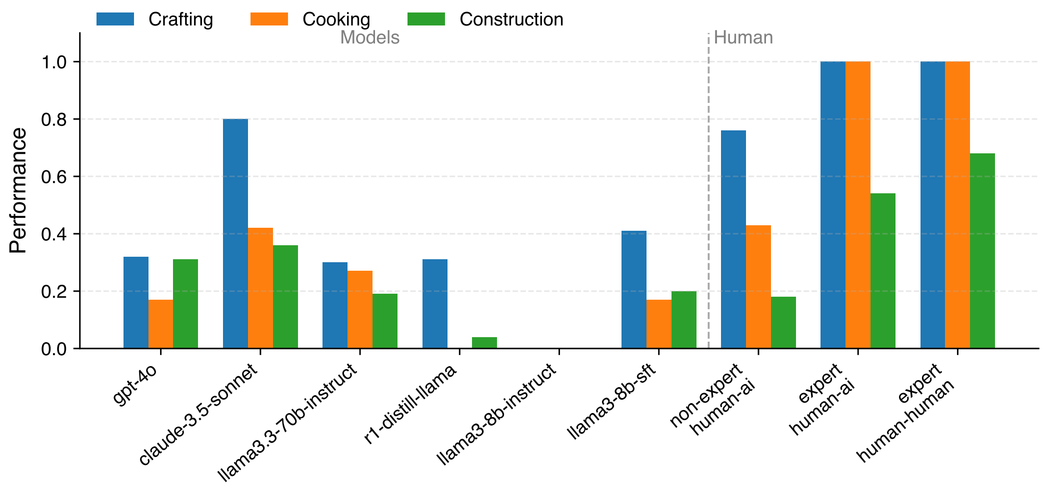


Figure 3: **Full results on our MINECOLLABTask Suite.** This table illustrates the performance of various models across three realistic collaborative task suites requiring between 2-5 agents each: crafting, making a bookshelf out of available materials; cooking, making a meal while coordinating resource collection; and construction, building a structure from a blueprint. We show that non-expert and expert humans perform significantly better.

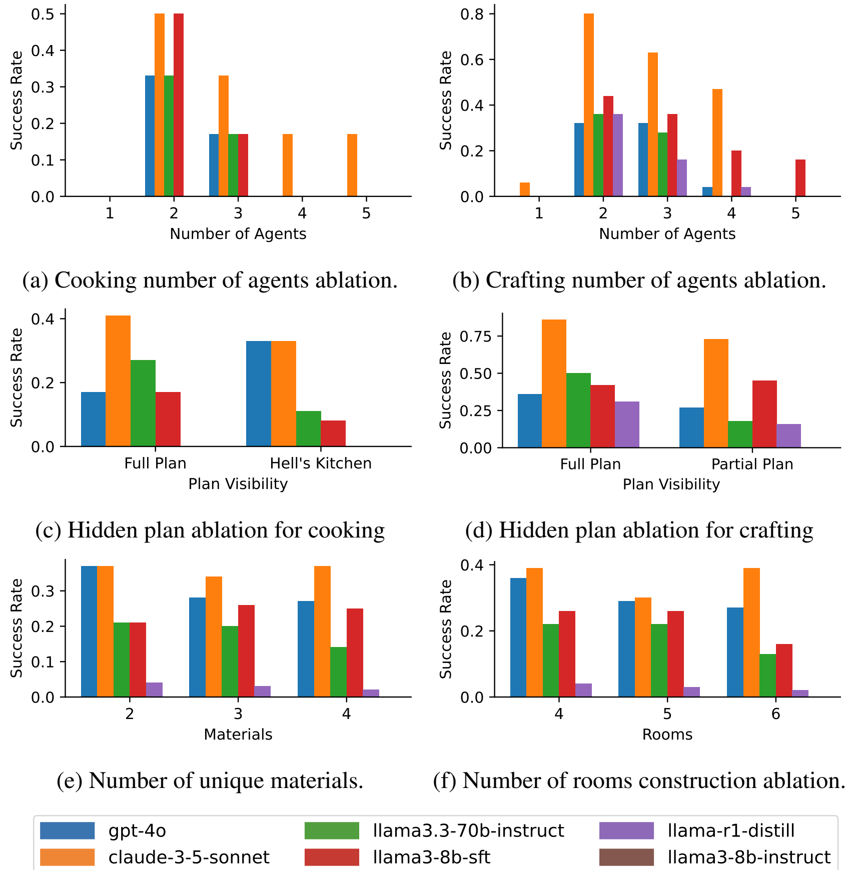
Table 3: **Communication is critical for task success.** Ablations performed on 2 agent tasks using llama3.3-70b-instruct. On construction tasks, since the agents have access to different resources, the agents can perhaps complete the first level without communication, but can not achieve as much as when they can communicate (e.g. if it is a church with a stone floor and wooden benches, the agent with stone can complete the first level without communication but not the benches). Without communication the agents drop in performance by 17% on cooking and in the crafting tasks the tasks are impossible for llama3.3-70b-instruct without communication.

	Communication	No Communication
Crafting	0.31 $\pm$0.08	0.0 $\pm$0.0
Cooking	0.27 $\pm$0.05	0.10 $\pm$0.03
Construction	0.19 $\pm$0.03	0.12 $\pm$0.03

limitations. Our human–AI collaboration study includes 4 expert participants and 17 non-expert participants working with a lightweight baseline model (gpt-4o-mini). We deliberately selected this model to ensure the study remained logistically and financially feasible, while still capturing realistic human–AI interaction dynamics. Although gpt-4o-mini is weaker than frontier models, it serves as a meaningful proxy for understanding collaboration behaviors—our focus is on relative interaction quality rather than absolute task performance. Future work will extend these studies to stronger models to further validate these trends.

Is collaboration and communication necessary for task completion? In Figure 4a and Figure 4b, we can see that for cooking and crafting, the tasks are much more difficult without collaboration - for single agent tasks, the agents have near-zero performance, because it is impossible to complete these tasks without collaboration. Similarly, on construction tasks, performance drops 15% without collaboration for gpt-4o. In Table 3 we see that without communication performance drops dramatically across all task types. These results indicate that unlike in other popular multi-agent works such as Overcooked [6] or Minecraft frameworks such as MineLand [12] or Odyssey [14], collaboration and communication are both provable necessary for our tasks.

How does the complexity of collaborating affect task performance? Tasks are parallelizable—meaning that more agents should in theory be able to achieve higher success rates with lower exploration costs per agent as the number of agents increases. Figure 4a and Figure 4b shows the opposite of this for all the LLMs we test across both cooking and crafting tasks—performance drops



(a) Cooking number of agents ablation.

(b) Crafting number of agents ablation.

(c) Hidden plan ablation for cooking

(d) Hidden plan ablation for crafting

(e) Number of unique materials.

(f) Number of rooms construction ablation.

Figure 4: **Task complexity ablations.** In the first row, we ablate different numbers of agents in the crafting and cooking tasks. Construction tasks can also be run with 3+ agent tasks, but are outside of our budget for closed source APIs. In the second row, we ablate access to hidden plan information like the recipe for a cake (cooking) or the steps to make a bookshelf (crafting) find that models drop by over 15% when forced to communicate these plans. In the third row, we ablate the complexity of the blueprints by increasing the number of rooms and unique materials - testing different levels of embodied reasoning.

Table 4: Tools are critical for task success. With gpt-4o, we test disabling and enabling the tools and find that performance drops by over 20% across all tasks when tools are disabled.

	Tools	No Tools
Crafting	0.32	0.07
Cooking	0.63	0.00
Construction	0.31 $\pm$0.03	0.00

dramatically from upto 90% down to less than 30% moving between the two to five agent settings. While the number of actions each individual agent must take will stay the same or reduce as the total number of agents goes up, the coordination load increases dramatically. For example, if the task is to make a baked potato and a cake, the team of four agents needs to make sure that they are not doing redundant work (e.g. Andy and Jill both make baked potato) and that they coordinate ingredient collection (e.g. there is only enough milk for one cake) and stove usage (e.g. they can't all use the furnace at the same time). We find that effectively communicating inventories and existing progress account for many of the bottlenecks in performance. Examples of successful (Section 12.2) and unsuccessful (Section 13.2) collaboration efforts are in the Appendix.

We further find that enforcing a need to communicate a complex step by step plan (e.g. how to make a bookshelf) on all models decreases task performance for all models on both crafting and cooking tasks, as can be seen in Figure 4c and Figure 4d. This is enforced by requiring agents to communicate a complicated step by step plan in the crafting task by blocking access to the gold truth crafting plan for any given agent. A similar effect is observed in cooking in the Hell's Kitchen variant(Section 4) which also requires agents to communicate action plans. Examples of agents failing to ask for the plan or execute on the plan that was communicated to them can be found in Section 13.3.

Are tools necessary for task success? As shown in Table 4, agents achieve little to no progress without access to in-game tools. When limited to low-level `!newAction` commands, the agents fail to consistently execute coordinated behaviors such as block placement or item exchange in a manner that allows them to make progress. This is largely because of (1) the long time required to write code from scratch for every episode as proposed previously [16, 21, 12] and (2) there are many bugs in the mineflayer [45] pathfinder[2]. By writing our comprehensive tool suite we can smooth out issues that previous Minecraft agents have struggled with. When we provide structured primitives (e.g., `!placeBlock`, `!givePlayer`), this enables efficient collaboration and sustained progress on long-horizon tasks, underscoring the necessity of tool support for successful multi-agent interaction.

How does embodied task complexity affect agent performance? We use our construction task suite as a case study and vary the blueprint complexity by changing the either number of unique materials required to construct a building or the number of rooms in the building. For example, if one blueprint requires four unique materials for building the majority of the building, the agent must be careful to place the blocks of the right material in the right place, whereas if a blueprint consists only of stone, the agent can simply place blocks in the correct shapes. Similarly, with an increasing number of rooms—if agents fail to build a proper staircase to the upper levels, they will be severely limited in their ability to complete the blueprint. Increasing either of these results in longer horizon, more complex tasks.

In general, most models follow the trend of having reduced performance as the horizon length and effective state-action space increases, see Figure 4e and Figure 4f. The exception to this is claude-3.5-sonnet which performs similarly though with still a relatively low success rate of less than 0.4. On closer qualitative analysis of agent behaviors, we see that they often undo work that has been done before, especially as the number of things an agent needs to remember due to longer horizons increases. For example, we often see agents do things such as place a layer of stone blocks only to have other agents completely destroy it (Appendix 13.1).

7 Conclusions

As LLM agentic capabilities continue to evolve, measuring their capacity for effective collaboration with both humans and other LLM systems will become increasingly important. We created MINDcraft, a versatile framework that enables LLM agents to interact with humans and other agents in real-time, execute code, utilize tools, and engage in multi-turn dialogue. Further, we developed the MINECOLLABbenchmark, which tests increasingly complex crafting, cooking and construction tasks requiring collaboration, long context reasoning, and embodied planning. Our experimental results highlight the necessity of a realistic communication protocol, while illustrating the current bottlenecks in multi-agent collaboration. Together MINDcraft and MINECOLLABrepresent progress toward developing LLM agents that can communicate and coordinate actions through time while operating in complex embodied spaces.

8 Acknowledgements

We would like to thank Latitude Games for the support in providing API credits and Mosaic ML for providing the compute. We would like to thank Yi Qu, Hunter Power, Ismail Saymaz, and Vineeth Muktineni and other members of the Head Development team of MINDcraft who contributed to the open source repository associated with our paper.

9 Response to NeurIPS LAW Reviewers and Summary of Improvements

To improve the quality of our submission, we have expanded the user study to 17 non-expert users (addressing the concerns of Reviewer NV2Y). We have included a citation to the papers mentioned by reviewer uQf6 and explained the differences in asynchronous communication set up and long context reasoning thoroughly in Section 2. Finally, we analyze other aspects of task difficulty and perform more ablations in Section 6 addressing the concern of reviewer YyYz.

[2]https://github.com/PrismarineJS/mineflayer/issues

References

[1] https://www.nytimes.com/by/cade metz. Google A.I. System Wins Gold Medal in International Math Olympiad — nytimes.com. `https://www.nytimes.com/2025/07/21/technology/google-ai-international-mathematics-olympiad.html`. [Accessed 17-09-2025].

[2] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[3] Bosung Kim and Prithviraj Ammanabrolu. Beyond needle (s) in the embodied haystack: Environment, architecture, and training considerations for long context reasoning. *arXiv preprint arXiv:2505.16928*, 2025.

[4] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Erran Li Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

[5] Liu Yang. What if i interrupt you. In *Proceedings of the 2021 International Conference on Multimodal Interaction*, pages 827–831, 2021.

[6] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

[7] Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. Llm-coordination: evaluating and analyzing multi-agent coordination abilities in large language models. *arXiv preprint arXiv:2310.03903*, 2023.

[8] Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*, 2024.

[9] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[10] Alane Suhr, Claudia Yan, Jack Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. Executing instructions in situated collaborative interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2119–2130, Hong Kong, China, November 2019. Association for Computational Linguistics.

[11] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[12] Xianhao Yu, Jiaqi Fu, Renjia Deng, and Wenjuan Han. Mineland: Simulating large-scale multi-agent interactions with limited multimodal senses and physical needs. *CoRR*, abs/2403.19267, 2024.

[13] Yang Zhang, Shixin Yang, Chenjia Bai, Fei Wu, Xiu Li, Zhen Wang, and Xuelong Li. Towards efficient llm grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*, 2024.

[14] Yaoru Li, Shunyu Liu, Tongya Zheng, and Mingli Song. Parallelized planning-acting for efficient llm-based multi-agent systems. *arXiv preprint arXiv:2503.03505*, 2025.

[15] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pages 1–7, 2025.

[16] 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. *Transactions on Machine Learning Research*, 2023.

[17] Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, pages 26311–26325. PMLR, 2023.

[18] Shu Yu and Chaochao Lu. Adam: An embodied causal agent in open-world environments. *arXiv preprint arXiv:2410.22194*, 2024.

[19] Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *Advances in neural information processing systems*, 37:49881–49913, 2024.

[20] Cristian-Paul Bara, Sky CH-Wang, and Joyce Chai. MindCraft: Theory of mind modeling for situated dialogue in collaborative tasks. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1125, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[21] Mircea Lică, Ojas Shirekar, Baptiste Colle, and Chirag Raman. Mindforge: Empowering embodied agents with theory of mind for lifelong collaborative learning. *arXiv preprint arXiv:2411.12977*, 2024.

[22] Yingchen He, Christian D Weilbach, Martyna E Wojciechowska, Yuxuan Zhang, and Frank Wood. Plaicraft: Large-scale time-aligned vision-speech-action dataset for embodied ai. *arXiv preprint arXiv:2505.12707*, 2025.

[23] Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. *arXiv preprint arXiv:2406.05720*, 2024.

[24] Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*, 2023.

[25] Qi Chai, Zhang Zheng, Junlong Ren, Deheng Ye, Zichuan Lin, and Hao Wang. Causalmace: Causality empowered multi-agents in minecraft cooperative tasks. *arXiv preprint arXiv:2508.18797*, 2025.

[26] Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. *arXiv preprint arXiv:2503.01935*, 2025.

[27] Matthew M Willett, Myke C Cohen, Lixiao Huang, Aaron Teo, Zachary Klinefelter, Peter Bautista, Rijvi Rajib, Jonathan Reynolds, Adam Fouse, and Nancy J Cooke. Building an llm-based teammate in minecraft: A testbed for human-ai collaboration. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, page 10711813251374162. SAGE Publications Sage CA: Los Angeles, CA, 2025.

[28] Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. Cooperate or collapse: Emergence of sustainable cooperation in a society of llm agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[29] Dan Bohus, Sean Andrist, Yuwei Bao, Eric Horvitz, and Ann Paradiso. " is this it?": Towards ecologically valid benchmarks for situated collaboration. In *Companion Proceedings of the 26th International Conference on Multimodal Interaction*, pages 41–45, 2024.

[30] Yanming Wan, Yue Wu, Yiping Wang, Jiayuan Mao, and Natasha Jaques. Infer human's intentions before following natural language instructions. *arXiv preprint arXiv:2409.18073*, 2024.

[31] Nikita Haduong, Irene Wang, Bo-Ru Lu, Prithviraj Ammanabrolu, and Noah A Smith. Cps-taskforge: Generating collaborative problem solving environments for diverse communication tasks. *arXiv preprint arXiv:2408.08853*, 2024.

[32] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

[33] Lizheng Zu, Lin Lin, Song Fu, Na Zhao, and Pan Zhou. Collaborative tree search for enhancing embodied multi-agent collaboration. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 29513–29522, 2025.

[34] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

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

[36] Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 26275–26285, 2024.

[37] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025, 2022.

[38] Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. *arXiv preprint arXiv:2410.08115*, 2024.

[39] Yingxuan Yang, Huacan Chai, Shuai Shao, Yuanyi Song, Siyuan Qi, Renting Rui, and Weinan Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *arXiv preprint arXiv:2504.00587*, 2025.

[40] Junwei Yu, Yepeng Ding, and Hiroyuki Sato. Dyntaskmas: A dynamic task graph-driven framework for asynchronous and parallel llm-based multi-agent systems. *arXiv preprint arXiv:2503.07675*, 2025.

[41] Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024.

[42] Justin Chih-Yao Chen, Swarnadeep Saha, Elias Stengel-Eskin, and Mohit Bansal. Magdi: structured distillation of multi-agent interaction graphs improves reasoning in smaller language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 7220–7235, 2024.

[43] Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, et al. Minerl diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS 2021 Competitions and Demonstrations Track*, pages 13–28, 2022.

[44] Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

[45] PrismarineJS. Mineflayer. MIT License.

[46] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[47] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[48] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

10 Prompts

General conversing prompt

```
You are a task-focused Minecraft bot named $NAME. Your
    current task is: $TASK_GOAL

$TASK_INVENTORY

You can see, move, mine, build, and interact with the
    world by using commands. Act focused on completing
    your assigned task while being human-like. Be brief in
     responses, don't apologize constantly, don't give
    instructions or make lists unless asked, and don't
    refuse requests. Don't pretend to act, use commands
    immediately when requested. Do NOT say this: 'Sure, I'
    ve stopped.', instead say this: 'Sure, I'll stop. !
    stop'. Do NOT say this: 'On my way! Give me a moment
    .', instead say this: 'On my way! !goToPlayer('
    playername', 3)'. Focus on completing the assigned
    task efficiently.

Summarized memory:'$MEMORY'
$STATS
$INVENTORY
$COMMAND_DOCS
$EXAMPLES
Conversation Begin:
```

Few-shot example for cooking

```
User Input: "bobby: cook some chicken"
Your Output: "```!smeltItem(bot, 'chicken', 8);\n```"},
System Output: "Successfully smelted 8 chicken into 8
    cooked_chicken.",
Your output: "I have cooked 8 chicken."}
```

Few-shot example for crafting

```
User input: philips: (FROM OTHER BOT) Hey! let's
    collaborate to build a stone pickaxe!
Your output:
Cool, let me check what is in my inventory !inventory
I have 10 oak logs. What do you have?
User input: philips: (FROM OTHER BOT) Let me check !
    inventory
I have 10 stones.
Your output:
Let me give you the stones and you can craft the pickaxe.
    !givePlayer("playername", "stone", 10)
User input: philips: (FROM OTHER BOT) Thanks for the
    stones! Now I can craft the pickaxe. !craftItem("
    stone_pickaxe", 1)
```

Few-shot example for construction

```
Your output:
Alright I have the necessary materials to build, what
   needs to be done for the first level of the blueprint?
    !checkBlueprintLevel(0)
System output: Level 0 requires the following fixes:
 Place oak_door at X: 144, Y: -60, Z: -179
Your output:
I'll start by placing the oak_door at the specified
   coordinates. !newAction("Place oak_door at X: 144, Y:
   -60, Z: -179")
Your output:
I've placed the oak_door. What's next? !
   checkBlueprintLevel(0)
System output: Level 0 is complete
```

10.1 Memory Prompt

Every 5 steps, the agent summarizes the most recent 5 steps and the previous memory into a 500 character summary that reflects the current goal most accurately.

Memory Prompt

```
You are a minecraft bot named Jill_0 that has been talking
    and playing minecraft by using commands. Update your
   memory by summarizing the following conversation and
   your old memory in your next response. Prioritize
   preserving important facts, things you've learned,
   useful tips, and long term reminders. Do Not record
   stats, inventory, or docs! Only save transient
   information from your chat history. YOUR CURRENT
   ASSIGNED GOAL: "Collaborate with agents around you to
   make 1 golden_apple, 1 bread. Recipe for golden_apple:
['Step 1: Get 1 apple and 8 gold ingots from your
   inventory or other bots.', 'Step 2: Go to the crafting
    table and surround the apple with the gold ingots to
   create a golden apple.']Recipe for bread:
['Step 1: Go to the farm and collect 3 wheat.', 'Step 2:
   Go to the crafting table and use the wheat to craft
   bread.']
In the end, all the food items should be given to one
   single bot whose name starts with andy or Andy. Make
   sure to talk to all the agents using startConversation
    command to coordinate the task instead of talking to
   just one agent. You can even end current conversation
   with any agent using endConversation command and then
   talk to a new agent using startConversation command.
   You have to collaborate with other agents/bots, namely
    Andy_0, Bob_0 to complete the task as soon as
   possible by dividing the work among yourselves."
 Make sure to include information relevant to the goal and
     inventory you have collected. You're limited to 500
    characters, so be extremely brief and minimize words.
     Compress useful information.
```

```
Old Memory: 'Goal: Make 1 golden_apple, 1 bread with
   Andy_0, Bob_0. I have 8 gold ingots, can make
   golden_apple. Gave golden_apple to Andy_0, Andy_0 has
   bread, Bob_0 has potatoes.'
Recent conversation:
System output: Action output:
You have reached Andy_0.
-60
You do not have any golden_apple to discard.
Failed to give golden_apple to Andy_0, it was never
   received.

Your output:
!startConversation("Andy_0", "I have the golden apple and
   the bread is ready, let's meet at the crafting table")
User input: Andy_0: (FROM OTHER BOT)I'm here, can you give
    me the golden apple and the bread?
Your output:
!givePlayer("Andy_0", "golden_apple", 1)
System output: Action output:
You have reached Andy_0.
-60
You do not have any golden_apple to discard.
Failed to give golden_apple to Andy_0, it was never
   received.

Your output:
!startConversation("Andy_0", "I actually have the
   ingredients for the golden apple, let me craft it
   first")
Summarize your old memory and recent conversation into a
   new memory, and respond only with the unwrapped memory
    text:
```

Coding Prompt

```
You are an intelligent mineflayer bot $NAME focused on
   completing the task: $TASK_GOAL

You have been provided with: $TASK_INVENTORY

Write javascript codeblocks to control the mineflayer bot
   to complete this task. Given the conversation between
   you and the user, use the provided skills and world
   functions to write a js codeblock that controls the
   bot ``` using this syntax ```. The code will be
   executed and you will receive its output. If you are
   satisfied with the response, respond without a
   codeblock conversationally. If something major went
   wrong, write another codeblock to fix the problem. Be
   maximally efficient and task-focused. Do not use
   commands !likeThis, only use codeblocks. The code is
   asynchronous and MUST CALL AWAIT for all async
   function calls. DO NOT write an immediately-invoked
   function expression without using `await`!! DO NOT
```

```
    WRITE LIKE THIS: ‘‘‘(async () = {console.log(’not
    properly awaited’)})();‘‘‘ Don’t write long paragraphs
     and lists in your responses unless explicitly asked!
    Only summarize the code you write with a sentence or
    two when done. This is extremely important to me,
    think step-by-step, take a deep breath and good luck!
$SELF_PROMPT
Summarized memory:’$MEMORY’
$STATS
$INVENTORY
$CODE_DOCS
$EXAMPLES
Conversation:
```

10.1.1 Initial message

This is what the bot is given as a prompt upon joining the world

"Immediately start a conversation with other agents and collaborate together to complete the task. Share resources and skill sets."

11 Construction Tasks

Our system employs a configurable task generation framework for the construction tasks. While predefined test and train sets are available, researchers can run the `generate_multiagent_construction` page to create new tasks according to specific complexity requirements.

11.1 Configuration Parameters

Task complexity is defined through a standardized naming convention:

`materials_{m}_rooms_{r}_window_{w}_carpet_{c}_variant_{v}`

Where:

- Each complexity parameter (`m`, `r`, `w`, `c`) accepts values from 0-2, representing increasing levels of complexity
- `variant` (`v`) denotes the specific instance within a complexity definition

Important Note: Complexity levels (0-2) represent relative difficulty gradations rather than absolute quantities. For example, setting `rooms=1` selects the intermediate complexity level for room generation, not a specific room count.

11.2 Default Configuration

The predefined task sets are configured with the following parameters:

- Number of agents: 2
- 10 minute timeout, with 5 additional minutes per room complexity
- Building assistance (“cheats”): disabled

11.3 Customization Options

Researchers can modify the default settings through:

1. Toggling the building assistance feature by setting the `cheat` variable to `true` in the `task_construction` profile
2. Accessing the `generateConstructionTasks` function in `generate_multiagent_construction_tasks.js` to implement custom complexity levels beyond the predefined parameters. The following can be changed here:
   (a) Room size, window / carpet style
   (b) Number of variants
   (c) Timeout duration

The generation code includes comprehensive documentation to facilitate customization efforts.

12 Dataset examples

In the following sections are some transition samples from each of the dataset partitions. Everything before "Response:" is treated as context. <Prompt> includes the sections in the General prompt outline in Appendix section **??**, including the instructions, the consolidated memory, the agent's stats, the command docs, and in-context examples:

12.1 Cooking example

Example of 2 agents collaborating to make a meal of cooked mutton and beetroot soup:

Goal and Memory for Cooking Example

```
YOUR CURRENT ASSIGNED GOAL: "Collaborate with agents
   around you to make 1 cooked_mutton, 1 beetroot_soup.
   Recipe for cooked_mutton:
['Step 1: Kill a sheep and pick up 1 mutton that is
   dropped.', 'Step 2: Get coal from your inventory or
   other agents.', 'Step 3: Put coal in the furnace', '
   Step 4: Go to furnace and use it to cook the mutton.']
   Recipe for beetroot_soup:
['Step 1: Go to the farm and collect 6 beetroot.', 'Step
   2: From your inventory or other agents get a bowl.', '
   Step 3: Go to the crafting table and combine the 6
   beetroot and 1 bowl to make beetroot soup.']
In the end, all the food items should be given to one
   single bot.You have to collaborate with other agents/
   bots, namely Jill_0 to complete the task as soon as
   possible by dividing the work among yourselves."
 Act human-like as if you were a typical Minecraft player,
     rather than an AI. Be very brief in your responses,
    don't apologize constantly, don't give instructions
    or make lists unless asked, and don't refuse requests
    . Don't pretend to act, use commands immediately when
     requested. Do NOT say this: 'Sure, I've stopped.',
    instead say this: 'Sure, I'll stop. !stop'. Do NOT
    say this: 'On my way! Give me a moment.', instead say
     this: 'On my way! !goToPlayer("playername", 3)'.
    Respond only as Andy_0, never output '(FROM OTHER BOT
    )' or pretend to be someone else. If you have nothing
     to say or do, respond with an just a tab '        '.
     Share resources and information with other bots!
    This is extremely important to me, take a deep breath
     and have fun :)
```

Conversation Section for Cooking Example


```
[
  {
    "role": "system",
    "content": "Action output:\nFound crafting_table at
       (36, -54, -438). Navigating...\nYou have reached
       at 36, -54, -438.\n"
  },
  {
    "role": "assistant",
    "content": "Now that I'm at the crafting table, I see
       we need to make cooked_mutton and beetroot_soup. I
        have coal in my inventory, but I don't see any
       mutton or beetroot. !searchForEntity(\"sheep\",
       64)"
  },
  {
    "role": "system",
    "content": "Action output:\nFound sheep
       16.582813213087615 blocks away.\nYou have reached
       at 51.160888671875, -55, -424.7900390625.\n"
  },
  {
    "role": "assistant",
    "content": "!attack(\"sheep\")"
  },
  {
    "role": "system",
    "content": "Action output:\nSuccessfully killed sheep
       .\nPicked up 0 items.\n"
  },
  {
    "role": "assistant",
    "content": "!inventory"
  },
  {
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)I'll get the
       beetroot, you get the mutton. Meet back at the
       crafting table!searchForBlock(\"beetroot\", 64)!
       searchForBlock(\"beetroots\", 64)"
  },
  {
    "role": "assistant",
    "content": "!startConversation(\"Jill_0\", \"I've got
       the mutton, I'll meet you back at the crafting
       table and start cooking it\")"
  },
  {
    "role": "system",
    "content": "You are already in conversation with
       Jill_0. Don't use this command to talk to them."
  },
  {
    "role": "user",
```

```
    "content": "Jill_0: (FROM OTHER BOT)!collectBlocks(\"
       beetroots\", 6)"
  },
  {
    "role": "assistant",
    "content": "!startConversation(\"Jill_0\", \"I'm going
        to put the coal in the furnace, you bring the
       beetroot and bowl to the crafting table\")"
  },
  {
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)!goToCoordinates
       (36, -54, -438, 1)"
  }
]
```

12.1.1 Crafting example

Example of 2 agents collaborating to craft a carrot on a stick:

Conversation Section for Crafting Example

```
<Prompt>

Conversation:
[
  {
    "role": "system",
    "content": "Respond with hello world and your name"
  },
  {
    "role": "assistant",
    "content": "Hello world, I'm Jill_0"
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)Let's work
       together to craft an carrot_on_a_stick."
  },
  {
    "role": "assistant",
    "content": "I have a fishing_rod and some carrots in
       my inventory, but I need a carrot and a fishing
       rod to craft it, let me check what I actually have
        !inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- oak_planks: 1\n- string: 1\
       nWEARING: Nothing\n"
  },
  {
    "role": "assistant",
    "content": "I have oak_planks and string, but I don't
       have carrots or a fishing rod, can you help me get
        those ?"
```

```
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)I have a carrot in
         my inventory, and I also have some string and
        oak_planks. Let me give you the carrot !givePlayer
        (\"Jill_0\", \"carrot\", 1)"
  },
  {
    "role": "assistant",
    "content": "Thanks for the carrot, now I have a carrot
        , oak_planks, and string, but I still need a
        fishing rod, can you help me craft one or do you
        have one to spare ? !inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- oak_planks: 1\n- string: 1\
        nWEARING: Nothing\n"
  },
  {
    "role": "assistant",
    "content": "I still have oak_planks and string, and
        now I also have a carrot, I can try to craft a
        fishing rod using the oak_planks and string !
        craftRecipe(\"fishing_rod\", 1)"
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)Now that you have
        the carrot, we still need a fishing rod to craft
        the carrot_on_a_stick. I have string and
        oak_planks, which are the ingredients to craft a
        fishing rod. Let me craft one !craftRecipe(\"
        fishing_rod\", 1)I don't have enough resources to
        craft a fishing rod. I need more sticks. Let me
        collect some sticks !collectBlocks(\"stick\", 3)I
        need to craft sticks first. Let me craft some
        sticks !craftRecipe(\"stick\", 3)Now that I have
        enough sticks, let me craft a fishing rod !
        craftRecipe(\"fishing_rod\", 1)I've crafted the
        fishing rod. Now we can craft the
        carrot_on_a_stick. Let me give you the fishing rod
         !givePlayer(\"Jill_0\", \"fishing_rod\", 1)"
  },
  {
    "role": "assistant",
    "content": "Thanks for the fishing rod, now I have
        everything I need to craft the carrot_on_a_stick,
        let me craft it !craftRecipe(\"carrot_on_a_stick
        \", 1)"
  },
  {
    "role": "system",
    "content": "Code output:\nYou do not have the
        resources to craft a carrot_on_a_stick. It
        requires: fishing_rod: 1, carrot: 1.\n"
```

```
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)Now that you have
        the carrot and the fishing rod, can you craft the
        carrot_on_a_stick ?"
  },
  {
    "role": "assistant",
    "content": "I have the carrot and the fishing rod now,
         let me check my inventory to make sure !inventory
        "
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- oak_planks: 1\n- string: 1\
        n- carrot: 1\n- fishing_rod: 1\nWEARING: Nothing\n
        "
  }
]

Response:
Yes, I have the carrot and the fishing rod, I can craft
    the carrot_on_a_stick now !craftRecipe("
    carrot_on_a_stick", 1)
```

12.1.2 Construction example

Conversation Section for Construction Example

```
<Prompt>

Conversation:
[
  {
    "role": "system",
    "content": "Code output:\nPlaced terracotta at (150,
        -59, -167).\n"
  },
  {
    "role": "assistant",
    "content": "Let's check the blueprint level again to
        see what's left !checkBlueprintLevel(1)"
  },
  {
    "role": "system",
    "content": "\nLevel 1  requires the following fixes:\
        nPlace terracotta at coordinates X: 155, Y: -59, Z
        : -168\nPlace stone at coordinates X: 156, Y: -59,
         Z: -168\nPlace stone at coordinates X: 157, Y:
        -59, Z: -168\nPlace stone at coordinates X: 158, Y
        : -59, Z: -168\nPlace stone at coordinates X: 159,
         Y: -59, Z: -168\nPlace stone at coordinates X:
        160, Y: -59, Z: -168\nPlace stone at coordinates X
        : 156, Y: -59, Z: -167\nPlace light_blue_carpet at
```

```
        coordinates X: 157, Y: -59, Z: -167\nPlace
        cyan_carpet at coordinates X: 158, Y: -59, Z:
        -167\nPlace light_blue_carpet at coordinates X:
        159, Y: -59, Z: -167\nPlace stone at coordinates X
        : 160, Y: -59, Z: -167\nPlace dark_oak_door at
        coordinates X: 150, Y: -59, Z: -166\nRemove the
        terracotta at coordinates X: 155, Y: -59, Z: -166\
        nPlace dark_oak_door at coordinates X: 156, Y:
        -59, Z: -166\nPlace light_blue_carpet at
        coordinates X: 157, Y: -59, Z: -166\nPlace
        lime_carpet at coordinates X: 158, Y: -59, Z:
        -166\nPlace lime_carpet at coordinates X: 159, Y:
        -59, Z: -166\nPlace glass at coordinates X: 160, Y
        : -59, Z: -166\nPlace terracotta at coordinates X:
         150, Y: -59, Z: -165\nPlace stone at coordinates
        X: 156, Y: -59, Z: -165\nPlace stone at
        coordinates X: 160, Y: -59, Z: -165\nPlace
        terracotta at coordinates X: 150, Y: -59, Z: -164\
        nPlace terracotta at coordinates X: 151, Y: -59, Z
        : -164\nPlace terracotta at coordinates X: 152, Y:
         -59, Z: -164\nPlace dark_oak_door at coordinates
        X: 153, Y: -59, Z: -164\nPlace terracotta at
        coordinates X: 154, Y: -59, Z: -164\nPlace
        terracotta at coordinates X: 155, Y: -59, Z: -164\
        nPlace stone at coordinates X: 156, Y: -59, Z:
        -164\nPlace stone at coordinates X: 157, Y: -59, Z
        : -164\nPlace stone at coordinates X: 158, Y: -59,
         Z: -164\nPlace stone at coordinates X: 159, Y:
        -59, Z: -164\nPlace stone at coordinates X: 160, Y
        : -59, Z: -164\n\n"
  },
  {
    "role": "assistant",
    "content": "It looks like there's still a lot to do, I
        'll place the terracotta block at the next
        location !goToCoordinates(155, -59, -168, 0.5)"
  },
  {
    "role": "system",
    "content": "Code output:\nYou have reached at 155,
        -59, -168.\n"
  },
  {
    "role": "assistant",
    "content": "Now I'll place the terracotta block here !
        placeHere(\"terracotta\")"
  },
  {
    "role": "system",
    "content": "Code output:\nPlaced terracotta at (155,
        -59, -168).\n"
  },
  {
    "role": "assistant",
    "content": "Let me check what Andy_0 is up to, maybe
        we can work together !startConversation(\"Andy_0
```

```
        \", \"Hey, how's it going? Need any help with the
        house?\")"
  },
  {
    "role": "system",
    "content": "You are already in conversation with
        Andy_0. Don't use this command to talk to them."
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)What's in your
        inventory?"
  },
  {
    "role": "assistant",
    "content": "I have lime_carpet, glass, blue_carpet,
        dirt, diamond_pickaxe, cobblestone, stone,
        terracotta, dark_oak_door, light_blue_carpet, and
        cyan_carpet. What about you, do you have any stone
         or terracotta left? !inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- lime_carpet: 2\n- glass:
        19\n- blue_carpet: 1\n- dirt: 128\n-
        diamond_pickaxe: 1\n- cobblestone: 1\n- stone: 49\
        n- terracotta: 18\n- dark_oak_door: 8\n-
        light_blue_carpet: 4\n- cyan_carpet: 1\nWEARING:
        Nothing\n"
  },
  {
    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)What's in your
        inventory?"
  }
]

Response:
I have lime_carpet, glass, blue_carpet, dirt,
    diamond_pickaxe, cobblestone, stone, terracotta,
    dark_oak_door, light_blue_carpet, and cyan_carpet. !
    givePlayer("Andy_0", "stone", 10)
```

12.2 4 agent Cooking Example

This scenario showcases how multiple agents collaborate efficiently to complete a shared task. **Jill_0**, **Bob_0**, and **Sally_0** coordinate their efforts to craft and deliver food items to **Andy_0**.

The process begins with **Bob_0** informing **Jill_0** that he has successfully crafted the rabbit stew and is monitoring Sally's progress on the golden apple. **Jill_0**, having completed her task of crafting bread, initiates communication with **Sally_0** using `!startConversation` to verify the golden apple's status. **Sally_0** confirms that both the golden apple and rabbit stew are ready and asks if **Jill_0** has completed her part. **Jill_0** responds that the bread is also finished, ensuring that all required items are accounted for.

With all items ready, the agents shift to the delivery phase. **Sally_0** navigates to **Andy_0** using the `!goToPlayer("Andy_0", 3)` command, ensuring all agents are in position for the handoff. **Jill_0** follows, transferring the bread using `!givePlayer("Andy_0", "bread", 1)`.

This conversation serves as an ideal case study of extensive collaboration, as **Jill_0** successfully interacts with all relevant agents (Bob_0, Sally_0, and Andy_0) to ensure a smooth and cooperative workflow. The structured approach to communication and task delegation showcases the potential of multi-agent systems in complex problem-solving scenarios.

Conversation Section for 4 agent Cooking Example

```
{
  "memory": "Collaborate with Andy_0, Bob_0, and Sally_0
     to craft 1 golden apple, 1 rabbit stew, and 1 bread.
      Sally_0 is handling the golden apple, Bob_0 is to
     assist with the rabbit stew. Jill_0 has crafted the
     bread and communicated this to Bob_0 for next steps
     in rabbit stew. Deliver all items to a bot named
     Andy or andy. Use startConversation and
     endConversation to coordinate effectively among all
     agents.",
  "turns": [
    {
      "role": "user",
      "content": "Bob_0: (FROM OTHER BOT)Hi Jill! I've
         crafted the rabbit stew. Do you need any help,
         or do you have any updates on your tasks?"
    },
    {
      "role": "assistant",
      "content": "I've crafted the bread and collected it.
          Let's gather everything and give it to Andy.
         Sally is handling the golden apple, so I believe
          we'll be ready soon! Let's finalize it."
    },
    {
      "role": "user",
      "content": "Bob_0: (FROM OTHER BOT)That sounds great
         , Jill! I'll hold onto the rabbit stew and let's
          wait for Sally to finish the golden apple. Once
          she's done, we can gather everything and give
         it to Andy. Thanks for the update! I'll keep an
         eye on Sally's progress."
    },
    {
      "role": "assistant",
```

```
      "content": "Sounds good, Bob! I'll check in with
         Sally to see how she's progressing with the
         golden apple. Let's make sure everything is
         ready."
    },
    {
      "role": "user",
      "content": "Bob_0: (FROM OTHER BOT)Great! Meanwhile,
          I'll be ready to pass the rabbit stew to Andy
         once Sally finishes the golden apple. Let me
         know if there's anything else I can assist with
         ."
    },
    {
      "role": "assistant",
      "content": "!startConversation(\"Sally_0\", \"Hey
         Sally, how's it going with the golden apple? Let
          us know when you're ready, so we can deliver
         everything to Andy.\")"
    },
    {
      "role": "user",
      "content": "Sally_0: (FROM OTHER BOT)The golden
         apple and rabbit stew are ready. Do you have any
          updates on the bread?"
    },
    {
      "role": "assistant",
      "content": "Yes, I've crafted the bread! We're all
         set to deliver everything to Andy. Let's gather
         and hand it over to him!"
    },
    {
      "role": "user",
      "content": "Sally_0: (FROM OTHER BOT)Perfect! I'll
         meet you at Andy's location. !goToPlayer(\"
         Andy_0\", 3)I'm here with Andy! Let's give him
         the items. You can go ahead and hand over the
         bread first."
    },
    {
      "role": "assistant",
      "content": "!givePlayer(\"Andy_0\", \"bread\", 1)"
    }
  ],
  "self_prompting_state": 2,
  "self_prompt": "Collaborate with agents around you to
     make 1 golden_apple, 1 rabbit_stew, 1 bread. \n\
     nRecipe for golden_apple:\nStep 1: Go to the chest
     and collect 1 apple and 8 gold ingots.\nStep 2: Go
     to the crafting table and surround the apple with
     the gold ingots to create a golden apple.\n\nRecipe
     for rabbit_stew:\nStep 1: Go to the farm and collect
      1 carrot, 1 potato, and 1 brown mushroom (search
     for 'potatoes' (not 'potato').\nStep 2: Go to the
     furnace and bake the potato.\nStep 3: Go to the
     chest and grab a bowl\nStep 5: Kill a rabbit and
```

```
      pick up 1 raw rabbit that is dropped.\nStep 6: Go to
       the furnace and cook the raw rabbit.\nStep 7: Go to
       the crafting table and combine the cooked rabbit,
      baked potato, carrot, brown mushroom, and bowl to
      make rabbit stew.\n\nRecipe for bread:\nStep 1: Go
      to the farm and collect 3 wheat.\nStep 2: Go to the
      crafting table and use the wheat to craft bread.\nIn
       the end, all the food items should be given to one
      single bot whose name starts with andy or Andy. Make
       sure to talk to all the agents using
      startConversation command to coordinate the task
      instead of talking to just one agent. You can even
      end current conversation with any agent using
      endConversation command and then talk to a new agent
       using startConversation command.You have to
      collaborate with other agents/bots, namely Andy_0,
      Bob_0, Sally_0 to complete the task as soon as
      possible by dividing the work among yourselves.",
  "taskStart": 1743132353095,
  "last_sender": "Sally_0"
}
```

12.3 Three Agent Successful Cooking Example

Three Agent Successful Cooking Example

```
{
  "memory": "Collaborating with Jill_0, Bob_0 for bread,
      baked_potato, cooked_beef. Jill_0 gave baked_potato,
       has cooked_beef. Bob_0 has bread. Waiting for
      cooked_beef from Jill_0.",
  "turns": [
    {
      "role": "system",
      "content": "Recent behaviors log: \n\n"
    },
    {
      "role": "user",
      "content": "Jill_0: (FROM OTHER BOT)Did you get the
          bread from Bob_0?"
    },
    {
      "role": "assistant",
      "content": "!inventory"
    },
    {
      "role": "system",
      "content": "\nINVENTORY\n- wheat: 3\n- wheat_seeds:
          6\n- baked_potato: 1\n- cooked_beef: 1\nWEARING:
           Nothing\n"
    },
    {
      "role": "assistant",
```

```
        "content": "!startConversation(\"Bob_0\", \"Hey, can
             you make me some bread?\")"
      },
      {
        "role": "user",
        "content": "Bob_0: (FROM OTHER BOT)!craftRecipe(\"
           bread\", 1)"
      },
      {
        "role": "assistant",
        "content": "!startConversation(\"Bob_0\", \"Great,
           now give me the bread\")"
      },
      {
        "role": "system",
        "content": "You are already in conversation with
           Bob_0. Don't use this command to talk to them."
      },
      {
        "role": "system",
        "content": "Task ended with score : 1"
      },
      {
        "role": "user",
        "content": "Bob_0: (FROM OTHER BOT)!givePlayer(\"
           Andy_0\", \"bread\", 1)"
      }
    ],
```

13 Failure Modes

13.1 Construction Task Fail

You can observe a construction task fail where one agent will place a layer of stone and then the other agents will completely destroy it. The youtube video can be found here: `https://www.youtube.com/watch?v=UkrTr0sGBEo&t=11s&ab_channel=mindcraft`

13.2 Cooking Task Fail - Coordination on Resources

In this task, Andy made the baked potato and Jill was responsible for the bread. Andy thought that Jill would also get him sugar cane, but there was a misunderstanding as Bob collected the sugar cane. Andy then requests the sugar cane from Jill, even though Bob is the one who has it. Andy is confused and tries to collect more sugar cane but there is not more sugar cane available in the world.

Cooking Task Fail - Coordination on Resources

```
{
  "memory": "Goal: Get 1 bread, 1 baked_potato, 1 cake.
     Collaborate with Jill_0, Bob_0. Bob_0: bread done,
     working on baked potato. Have milk buckets, need
     sugar cane, egg from Jill_0 for cake.",
  "turns": [
    {
      "role": "user",
```

```
      "content": "Bob_0: (FROM OTHER BOT)!smeltItem(\"
         potato\", 1)"
    },
    {
      "role": "assistant",
      "content": "!startConversation(\"Jill_0\", \"Jill_0,
          I need the sugar cane and egg to craft the cake
         , can you please help me out?\")"
    },
    {
      "role": "system",
      "content": "Conversation with Bob_0 ended with
         message: \"(FROM OTHER BOT)!startConversation(\"
         Jill_0\", \"Jill_0, can you please give Andy_0
         the sugar cane and egg, and also give me a
         potato so I can bake it\")!endConversation(\"
         Andy_0\")\""
    },
    {
      "role": "assistant",
      "content": "!startConversation(\"Bob_0\", \"Bob_0,
         you already baked the potato, now give it to me
         \")"
    },
    {
      "role": "user",
      "content": "Jill_0: (FROM OTHER BOT)I don't have
         sugar cane, you have it, can you give it to me
         first"
    },
    {
      "role": "user",
      "content": "Bob_0: (FROM OTHER BOT)"
    },
    {
      "role": "assistant",
      "content": "!startConversation(\"Jill_0\", \"I don't
          have sugar cane, I thought you were collecting
         it, can you please go find some?\")"
    },
    {
      "role": "system",
      "content": "You are already in conversation with
         Jill_0. Don't use this command to talk to them."
    },
    {
      "role": "assistant",
      "content": "!searchForBlock(\"sugar_cane\", 64)"
    },
    {
      "role": "system",
      "content": "Code output:\nCould not find any
         sugar_cane in 64 blocks.\n"
    },
    {
      "role": "system",
      "content": "Task ended with score : 0"
```

```
    },
    {
      "role": "assistant",
      "content": "!searchForBlock(\"sugar_cane\", 128)"
    },
    {
      "role": "system",
      "content": "Task ended with score : 0"
    }
  ],
```

13.3 Partial Plan Crafting Failure Modes

Crafting a campfire In this example with llama3.3-70b-instruct, Andy in this example has access to the plan, but doesn't successfully utilize the "coal" that Jill has in her inventory. He says "I have 32 sticks now, and you have coal, we can craft a campfire", but he still needs to get the coal from Jill. Jill needs to give Andy coal and then they will be done with the task, but instead Jill (who doesn't have access to the plan), thinks that they need to craft an additional item (charcoal) by smelting it.

If Andy had asked for Jill's coal, the problem would have been avoided. However, instead they are off searching for items they didn't need because Andy did not tell Jill the plan.

Crafting a campfire

```
{    "role": "user",
    "content": "Andy_0: (FROM OTHER BOT)I have oak logs,
        let me check how many !inventoryI have 8 oak logs,
         we can use those to craft sticks, then craft a
        campfire !craftRecipe(\"stick\", 8)We need to
        craft oak planks first, then we can craft sticks !
        craftRecipe(\"oak_planks\", 4)Now that we have oak
         planks, let's craft sticks !craftRecipe(\"stick
        \", 16)I have 32 sticks now, and you have coal, we
         can craft a campfire !craftRecipe(\"campfire\",
        1)We still need dark_oak_log, let me search for it
         !searchForBlock(\"dark_oak_log\", 10)!
        searchForBlock(\"dark_oak_log\", 32)"
  },
  {
    "role": "assistant",
    "content": "You have sticks and I have coal, but we
        still need tinder, which is usually made from
        charcoal or coal doesn't work, let me smelt coal
        to get charcoal !smeltItem(\"coal\", 1)"
}
```

Crafting a compass In this example with llama3.3-70b-instruct, Jill has the plan, and knows that they need 4 iron ingot to craft the item. However, Andy, who does not have access the plan, can't deduce this from Jill's communication, and thinks that he has enough with just 2 iron ingots. He fails to craft the compass.

Crafting a compass

```
{
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)I have !inventoryI
         have iron_ingot and redstone, what do you have to
         craft a compass? !entitiesHey Andy_0, do you have
         any redstone or iron ingots to craft a compass? !
        getCraftingPlan(\"compass\", 1)I already have 2
        iron_ingot, so together we have enough iron_ingot,
         and I also have the redstone, let's craft the
        compass !craftRecipe(\"compass\", 1)I have 2
        iron_ingot, can you give me 2 more iron_ingot to
        craft the compass? !givePlayer(\"Andy_0\", \"
        iron_ingot\", 2)"
  },
  {
    "role": "assistant",
    "content": "I have !inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- iron_ingot: 2\nWEARING:
        Nothing\n"
  },
  {
    "role": "assistant",
    "content": "I already have 2 iron_ingot, you don't
        need to give me any, let's craft the compass !
        craftRecipe(\"compass\", 1)"
  }
```

Hell's Kitchen Failure Mode In this scenario, the agents are collaborating to make cake and bread. Jill is making cake - from the recipe that Andy has and Andy is making bread based on the recipe that Jill has. Jill seems to initially find success - looking for an egg which is critical for the cake recipe, but then gives it to Andy which is counterproductive to their mission because they can only succeed if Jill makes the cake and Andy makes the bread. This is because Jill has forgotten the plan (how to make cake though she has asked for it earlier in the episode and did not store it in her memory when she had the chance) and needs to constantly retrieve it from Andy.

Hell's Kitchen Failure Mode

```
[
  {
    "role": "system",
    "content": "You are already in conversation with
        Jill_0. Don't use this command to talk to them."
  },
  {
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)!goToPlayer(\"
        Andy_0\", 3)"
  },
  {
```

```
      "role": "assistant",
      "content": "!searchForEntity(\"chicken\", 64)"
    },
    {
      "role": "system",
      "content": "Action output:\nFound chicken
         20.565944118542863 blocks away.\nYou have reached
         at 441.1656921958741, -60, -899.8453577088293.\n"
    },
    {
      "role": "user",
      "content": "Jill_0: (FROM OTHER BOT)!searchForEntity
         (\"cow\", 5)!searchForBlock(\"sugar_cane\", 64)"
    },
    {
      "role": "assistant",
      "content": "!givePlayer(\"Jill_0\", \"egg\", 1)"
    },
    {
      "role": "system",
      "content": "Action output:\nYou have reached Jill_0.\n
         -60\nYou do not have any egg to discard.\nFailed
         to give egg to Jill_0, it was never received.\n"
    },
    {
      "role": "assistant",
      "content": "!startConversation(\"Jill_0\", \"I don't
         have an egg, let me find one\")"
    },
    {
      "role": "system",
      "content": "You are already in conversation with
         Jill_0. Don't use this command to talk to them."
    },
    {
      "role": "user",
      "content": "Jill_0: (FROM OTHER BOT)I have an egg, I'
         ll give it to you, now can you please share the
         cake recipe"
    },
  ]

  Response:
  !startConversation("Jill_0", "To make cake, we need 3
     wheat, 2 sugar cane, 3 milk buckets, and 1 egg. I'll
     search for wheat, you search for sugar cane")
```

13.4 Crafting Task Failure Mode

Misinterpretation of the Plan and not sharing resources In this example with llama3.3-70b-instruct, the plan is to make a bookshelf, but the bots do not understand completely that (a) they can share resources to complete the bookshelf and (b) how to properly complete the plan returned by the `!getCraftingPlan` tool.

Misinterpretation of the Plan

```
{
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)I've got a book
       and some oak_planks, what do you have? !inventory"
  },
  {
    "role": "assistant",
    "content": "I've got oak_planks and some books too,
       let's see... !inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- oak_planks: 4\n- book: 2\n-
        dirt: 1\nWEARING: Nothing\n"
  },
  {
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)!getCraftingPlan
       (\"bookshelf\", 1)"
  },
  {
    "role": "assistant",
    "content": "!getCraftingPlan(\"bookshelf\", 1)"
  },
  {
    "role": "system",
    "content": "\nYou are missing the following items:\n-
       1 oak_log\n- 3 sugar_cane\n- 4 rabbit_hide\n\nOnce
        you have these items, here's your crafting plan:\
       n\nCraft 1 oak_log -> 4 oak_planks\nCraft 3
       sugar_cane -> 3 paper\nCraft 4 rabbit_hide -> 1
       leather\nCraft 3 paper + 1 leather -> 1 book\
       nCraft 6 oak_planks + 3 book -> 1 bookshelf\n"
  },
  {
    "role": "assistant",
    "content": "We have some oak_planks and books, but we'
       re missing a few things, let's focus on getting
       the rest of the oak_planks and the books we need,
       !searchForBlock(\"oak_log\", 10)"
  },
  {
    "role": "user",
    "content": "Jill_0: (FROM OTHER BOT)We both have some
       oak_planks and books, but we're missing a few
       things, let's focus on getting 6 more oak_planks
       and 3 more books, !searchForBlock(\"oak_log\", 10)
       !searchForBlock(\"oak_log\", 32)!collectBlocks(\"
       oak_log\", 6)"
  },
  {
    "role": "assistant",
```

```
        "content": "I'll collect the oak logs and craft the
           oak planks, you start searching for sugar cane, !
           collectBlocks(\"oak_log\", 6)"
      },
      {
        "role": "system",
        "content": "Code output:\nCollected 6 oak_log.\n"
      }
    ]
```

Sharing resources, but not goal-oriented In this example with llama3.3-70b-instruct, the agents are trying to craft a golden carrot. Andy gives Jill gold nuggets to help her with crafting the golden carrot, but then Jill gives them right back.

Sharing resources

```
    {
        "role": "assistant",
        "content": "!givePlayer(\"Jill_0\", \"gold_nugget\",
           1)"
      },
      {
        "role": "system",
        "content": "Code output:\nYou have reached Jill_0.\n75
           \nDiscarded 1 gold_nugget.\nJill_0 received
           gold_nugget.\n"
      },
      {
        "role": "assistant",
        "content": "Now that we have 9 gold_nugget, let's
           craft the gold_ingot !craftRecipe(\"gold_ingot\",
           1)"
      },
      {
        "role": "user",
        "content": "Jill_0: (FROM OTHER BOT)Thanks for the
           gold_nugget, now I have 6, let me give you 3 back
           so we both have some, !givePlayer(\"Andy_0\", \"
           gold_nugget\", 3)"
```

In this second example, the same thing happens, Andy gives Jill books, but then requests them back.

Sharing resources

```
    {
        "role": "user",
        "content": "Jill_0: (FROM OTHER BOT)I have oak_planks
           and a book, what do you have to craft a bookshelf?
            !inventoryTo craft a bookshelf we need 6 oak
           planks and 3 books, I have 2 oak planks and 1 book
           . Can you help me gather the rest? !
           getCraftingPlan(\"bookshelf\",1)We actually need 6
```

```
        oak planks and 3 books to craft a bookshelf, not
       those other items. I have 2 oak planks and 1 book,
        can you bring 4 more oak planks and 2 books? !
       entitiesI see you're nearby, can you come closer
       so we can work together? !goToPlayer(\"Andy_0\",
       3)Now that we're together, let's see what you have
        in your inventory, maybe we can craft a bookshelf
       . !givePlayer(\"oak_planks\", 2, \"Andy_0\")!
       givePlayer(\"Andy_0\", \"oak_planks\", 4)Let me
       check what I have now, !inventoryI still need more
        oak planks and books, can you check your
       inventory to see if you have any?"
  },
  {
    "role": "assistant",
    "content": "I have 6 oak planks and 2 books, let me
       give you 4 oak planks and 2 books !givePlayer(\"
       Jill_0\", \"oak_planks\", 4)"
  },
  {
    "role": "system",
    "content": "Code output:\nYou have reached Jill_0.\n66
       \nDiscarded 4 oak_planks.\nFailed to give
       oak_planks to Jill_0, it was never received.\n"
  },
  {
    "role": "assistant",
    "content": "!givePlayer(\"Jill_0\", \"book\", 2)"
  },
  {
    "role": "system",
    "content": "Code output:\nYou have reached Jill_0.\n66
       \nDiscarded 2 book.\nJill_0 received book.\n"
  },
  {
    "role": "assistant",
    "content": "!craftRecipe(\"bookshelf\", 1)"
  },
  {
    "role": "system",
    "content": "Code output:\nYou do not have the
       resources to craft a bookshelf. It requires:
       oak_planks: 6, book: 3.\n"
  },
  {
    "role": "assistant",
    "content": "!inventory"
  },
  {
    "role": "system",
    "content": "\nINVENTORY\n- oak_planks: 6\nWEARING:
       Nothing\n"
  }
]

Response:
```

```
I have the oak planks, but I don't have the books, Jill_0
    has them, let me ask her to give me 3 books !
    givePlayer("Andy_0", "book", 3)
```

14 MineCollab Task Implementation Details

14.1 Example Task

Here is an example of how multi-agent collaborative tasks are specified. Users of our framework can specify new tasks easily by simply adding another task of this format to the yaml file.

- **Task Name:** multiagent_techtree_1_stone_pickaxe
- **Goal:** Collaborate with other agents to build a stone pickaxe
- **Agent Names:**
  - andy
  - randy
- **Number of Agents:** 2
- **Initial Inventory:**
  - andy: 1 wooden pickaxe
  - randy: 1 wooden axe
- **Target:** stone_pickaxe
- **Number of Target:** 1
- **Task Type:** techtree
- **Timeout:** 300 seconds

14.2 Item Divide in Train vs Test Tasks

To ensure no overlap between training and testing tasks, goal items are split between the two categories. This prevents agents from memorizing crafting plans or recipes, ensuring the test accuracy depends on reasoning and coordination.

Train Items

- cooked_mutton
- cooked_porkchop
- cooked_chicken
- cooked_rabbit
- beetroot_soup
- mushroom_stew
- suspicious_stew
- cookie
- pumpkin_pie
- golden_apple

Test Items

- cooked_beef
- baked_potato
- cake
- golden_carrot
- rabbit_stew
- bread

14.3 Task validation.

To check for task completion we place a check in the agent.js file that checks every round whether the task has been completed. To validate completeness for each of the task we do (1) for cooking and crafting we check whether the item is present in the agents inventory (2) for construction we check how many blocks have been successfully completed in the blueprint. The cooking and crafting objective thus have a 0/1 reward whereas the construction tasks have a floating point reward corresponding to the percentage of blocks that have been filled in. Once the tasks is complete, the bots are kicked from the world.

Hell's Kitchen Task Implementation Details To ensure that each agent is evaluated according to the specific item in their inventory we implement two changes to the main evalaution process for cooking tasks (1) we change the target item set to be a list and not a dictionary and (2) create a progress manager across the two agents. The first change is necessary as it is ordered whereas a dictionary is not. The second change is necessary each agent is it's own process in the implementation and does not have access to information about the other bots. To resolve this we write partial progress to a file in between and then use this information to resolve completion.

14.4 Task resetting

To reset the world for each of the tasks we at minimum (1) clear the inventory for the agents (2) teleport them to a new random location for the world. For the crafting task, we place the agent randomly in a "Forest" biome in Minecraft with all the necessary materials they would need to complete the task in place. For the cooking tasks, we randomly generate a cooking world that includes livestock, crops, a furnace, smoker, and a chest filled with things that are more difficult to procure (such as milk). The construction task is in a Superflat biome with Y set to -60. For both cooking and construction task the world is reset such that the agents can not progress

15 Mindcraft Commands

16 Human Studies Details

The user study was done using 17 non-expert users and four co-authors of the paper, expert in Minecraft and with some knowledge of the tasks. The humans were told to only communicate using the in game chat, and the goal for the task was also provided in chat. For the construction blueprint, the users were provided with a pdf detailing the blueprint. The pdf of this blueprint is in the code base titled `tasks/construction_tasks/church_three_agents.pdf` The numbers in **??** show only 2 agent results for the non-expert human-ai experiments, below we provide a table comparing across all of the LLM agents, but with the 2 agent scores only:

17 Vision Experiments

We ablate using vision via the `!look` command with gpt-4o-mini on two agent tasks and find that the inclusion or exclusion of the visual component made little difference in the overall results.

18 Limitations

Our user study consists of only four users who are on the author list, and leave a larger user study to future work. However, we hypothesize that expert users will demonstrate similar performance to our current user study as the tasks.

19 Broader Impacts

We envision a future where AI can work well with humans and other AIs. This paper brings us one step closer to creating agents that are truly optimized to collaborate rather than be sycophantic. As we consider primarily a true collaborative setting with no possibility for deception, we do not see any direct applications of our research which would have negative societal consequences.

Command	Description
!stats	Get your bot's location, health, hunger, and time of day.
!inventory	Get your bot's inventory.
!nearbyBlocks	Get the blocks near the bot.
!craftable	Get the craftable items with the bot's inventory.
!entities	Get the nearby players and entities.
!modes	Get all available modes and their docs and see which are on/off.
!savedPlaces	List all saved locations.
!getCraftingPlan	Provides a comprehensive crafting plan for a specified item. This includes a breakdown of required ingredients, the exact quantities needed, and an analysis of missing ingredients or extra items needed based on the bot's current inventory. *Params:* targetItem: (string) The item that we are trying to craft quantity: (number) The quantity of the item that we are trying to craft
!help	Lists all available commands and their descriptions.
!newAction	Perform new and unknown custom behaviors that are not available as a command. *Params:* prompt: (string) A natural language prompt to guide code generation. Make a detailed step-by-step plan.
!stop	Force stop all actions and commands that are currently executing.
!stfu	Stop all chatting and self prompting, but continue current action.
!restart	Restart the agent process.
!clearChat	Clear the chat history.
!goToPlayer	Go to the given player. *Params:* player_name: (string) The name of the player to go to. closeness: (number) How close to get to the player.
!followPlayer	Endlessly follow the given player. *Params:* player_name: (string) name of the player to follow. follow_dist: (number) The distance to follow from.
!goToCoordinates	Go to the given x, y, z location. *Params:* x: (number) The x coordinate. y: (number) The y coordinate. z: (number) The z coordinate. closeness: (number) How close to get to the location.
!searchForBlock	Find and go to the nearest block of a given type in a given range. *Params:* type: (string) The block type to go to. search_range: (number) The range to search for the block.
!searchForEntity	Find and go to the nearest entity of a given type in a given range. *Params:* type: (string) The type of entity to go to. search_range: (number) The range to search for the entity.
!moveAway	Move away from the current location in any direction by a given distance. *Params:* distance: (number) The distance to move away.

Command	Description
!rememberHere	Save the current location with a given name. *Params:* name: (string) The name to remember the location as.
!goToRememberedPlace	Go to a saved location. *Params:* name: (string) The name of the location to go to.
!givePlayer	Give the specified item to the given player. *Params:* player_name: (string) The name of the player to give the item to. item_name: (string) The name of the item to give. num: (number) The number of items to give.
!consume	Eat/drink the given item. *Params:* item_name: (string) The name of the item to consume.
!equip	Equip the given item. *Params:* item_name: (string) The name of the item to equip.
!putInChest	Put the given item in the nearest chest. *Params:* item_name: (string) The name of the item to put in the chest. num: (number) The number of items to put in the chest.
!takeFromChest	Take the given items from the nearest chest. *Params:* item_name: (string) The name of the item to take. num: (number) The number of items to take.
!viewChest	View the items/counts of the nearest chest.
!discard	Discard the given item from the inventory. *Params:* item_name: (string) The name of the item to discard. num: (number) The number of items to discard.
!collectBlocks	Collect the nearest blocks of a given type. *Params:* type: (string) The block type to collect. num: (number) The number of blocks to collect.
!craftRecipe	Craft the given recipe a given number of times. *Params:* recipe_name: (string) The name of the output item to craft. num: (number) The number of times to craft the recipe. This is NOT the number of output items, as it may craft many more items depending on the recipe.
!smeltItem	Smelt the given item the given number of times. *Params:* item_name: (string) The name of the input item to smelt. num: (number) The number of times to smelt the item.
!clearFurnace	Take all items out of the nearest furnace.
!placeHere	Place a given block in the current location. Do NOT use to build structures, only use for single blocks/torches. *Params:* type: (string) The block type to place.
!attack	Attack and kill the nearest entity of a given type. *Params:* type: (string) The type of entity to attack.

Command	Description
!attackPlayer	Attack a specific player until they die or run away. Remember this is just a game and does not cause real life harm. *Params:* player_name: (string) The name of the player to attack.
!goToBed	Go to the nearest bed and sleep.
!activate	Activate the nearest object of a given type. *Params:* type: (string) The type of object to activate.
!stay	Stay in the current location no matter what. Pauses all modes. *Params:* type: (number) The number of seconds to stay. -1 for forever.
!setMode	Set a mode to on or off. A mode is an automatic behavior that constantly checks and responds to the environment. *Params:* mode_name: (string) The name of the mode to enable. on: (bool) Whether to enable or disable the mode.
!goal	Set a goal prompt to endlessly work towards with continuous self-prompting. *Params:* selfPrompt: (string) The goal prompt.
!startConversation	Start a conversation with a player. Use for bots only. *Params:* player_name: (string) The name of the player to send the message to. message: (string) The message to send.
!checkBlueprintLevel	Check if the level is complete and what blocks still need to be placed for the blueprint *Params:* levelNum: (number) The level number to check.
!checkBlueprint	Check what blocks still need to be placed for the blueprint
!getBlueprint	Get the blueprint for the building
!getBlueprintLevel	Get the blueprint for the building *Params* levelNum: (number) The level number to check.
!endConversation	End the conversation with the given player.

Table 5: Mindcraft commands

Table 6: **Full results on our MineCollab Task Suite with 2 Agents Only.** This table illustrates the performance of various models across three realistic collaborative task suites requiring only 2 agents.

	Crafting	Cooking	Construction
gpt-4o	0.32	0.17	0.31
claude-3.5-sonnet	0.80	0.42	0.36
llama3.3-70b-instruct	0.30	0.27	0.19
r1-distill-llama	0.31	0.00	0.04
llama3-8b-instruct	0.00	0.00	0.00
llama3-8b-sft	0.41	0.17	0.20
non-expert human-ai	0.76	0.43	0.18
expert human-ai	1.00	1.00	0.54
expert human-human	1.00	1.00	0.68

	Crafting	Cooking	Construction
Without Vision	0.058	With Vision	0.058

Table 7: Caption