

TeamCraft: A Benchmark for Multi-Modal Multi-Agent Systems in Minecraft

Qian Long¹, Zhi Li¹, Ran Gong¹, Ying Nian Wu¹, Demetri Terzopoulos¹, Xiaofeng Gao^{2†}

¹University of California, Los Angeles

²Amazon AGI

teamcraftbench@gmail.com

<https://teamcraft-bench.github.io/>

Abstract

Collaboration is a cornerstone of society. In the real world, human teammates make use of multi-sensory data to tackle challenging tasks in ever-changing environments. It is essential for embodied agents collaborating in visually-rich environments replete with dynamic interactions to understand multi-modal observations and task specifications. To evaluate the performance of generalizable multi-modal collaborative agents, we present TeamCraft, a multi-modal multi-agent benchmark built on top of the open-world video game Minecraft. The benchmark features 55,000 task variants specified by multi-modal prompts, procedurally-generated expert demonstrations for imitation learning, and carefully designed protocols to evaluate model generalization capabilities. We also perform extensive analyses to better understand the limitations and strengths of existing approaches. Our results indicate that existing models continue to face significant challenges in generalizing to novel goals, scenes, and unseen numbers of agents. These findings underscore the need for further research in this area. The TeamCraft platform and dataset are publicly available at

<https://github.com/teamcraft-bench/teamcraft>

1. Introduction

Developing collaborative skills is essential for embodied agents, as collaboration is a fundamental aspect of human intelligence [49]. In the AI community, multi-agent collaboration is frequently studied using grid-world environments [10, 17, 26, 31, 42, 44, 50, 52, 60, 64]. However, agents in these environments lack multi-modal understanding. By contrast, learning within visually-rich environments enables agents to develop useful representations of multi-agent dynamics [7, 20], as vision facilitates implicit communication, coordination, and collaborative execution [21, 22].

Learning vision-based, multi-task, multi-agent systems is a challenging objective that presents several difficulties. These systems must develop detailed scene understanding to handle the diverse visual appearances of scenes. The complexity is further heightened by the numerous combinations of task configurations, such as object spatial arrangements, goal configurations, arbitrary numbers of agents, and heterogeneous agent capabilities. Consequently, it is essential for multi-agent systems to acquire generalizable skills that can be effectively transferred across different settings.

An important step in addressing these challenges is to develop simulation systems that support multi-modal multi-agent learning. Recent advances in simulated environments have significantly facilitated progress in embodied vision-based systems [7, 9, 21, 43, 62]. Despite notable progress, these systems have several limitations: (1) many of them target one or two-agent scenarios [22, 37, 57], (2) they are often limited to indoor settings with a narrow range of tasks [44, 66], and (3) the task specifications are generally purely in text [30, 37], making it hard to specify subtle task differences accurately and efficiently.

To drive progress in this area, we have developed a comprehensive benchmark, named *TeamCraft*, that features procedurally generated large-scale datasets specifically designed for multi-modal multi-agent systems. This benchmark utilizes the widely acclaimed open-world video game Minecraft as an experimental platform to engage with the complex dynamics of multi-modal multi-agent interactions. Inspired by the work of [24], we also leverage multi-modal prompts as task specifications to guide agent interactions, as language often fails to effectively convey spatial information [4]. Our benchmark offers rich visual backgrounds, diverse object categories, complex crafting sequences, and varying task dynamics. These features enable systematic exploration of out-of-distribution generalization challenges for multi-modal, multi-task, multi-agent systems at scale. In particular, our benchmark evaluates a model’s ability to generalize to novel goal configurations, unseen number of agents, novel agent ca-

† This work does not relate to the author’s position at Amazon.

Table 1. Comparison with other benchmarks. *TeamCraft* features visual observation for multi-agent control with widely-varied tasks specified by multi-modal prompts, targeting various types of generalization essential for multi-agent teaming. **MM Spec.**: multi-modal task specification. **Observation**: V for visual observation and S for state-based observation. **MA**: multi-agent control, C for centralized and D for decentralized. **Interaction**: object interaction. **Tool**: tool use. **Generalization**: types of generalization targeted, E for generalization on novel environments or scenes, G for novel goals, A for novel numbers of agents. **# Variants**: number of task variants involved.

Benchmark	MM Spec.	3D	Observation	MA	Interaction	Tool	Generalization	# Agents	# Variants	# Demonstrations
ALFRED [47]	X	✓	V	X	✓	✓	E	1	2,600+	8,000+
FurnMove [21]	X	✓	V	CD	✓	X	E	2	30	X
Marlo [43]	X	✓	V	D	✓	X	X	4+	14	X
MineDojo [11]	X	✓	V	X	✓	✓	EG	1	3,000+	740,000+
MindAgent [17]	X	✓	VS	C	✓	✓	X	4+	39	X
Neural MMO 2.0 [51]	X	X	S	CD	✓	✓	EGA	128+	25+	X
Overcooked-AI [5]	X	X	VS	C	✓	✓	X	2	5	80
PARTNR [6]	X	✓	VS	CD	✓	✓	E	2	100,000+	100,000+
RoCoBench [37]	X	✓	S	CD	✓	✓	G	2	6	X
VIMA-Bench [24]	✓	✓	V	X	✓	✓	EG	1	1,000+	600,000+
Watch&Help [44]	X	✓	S	CD	✓	✓	EG	2	1,200+	6,300+
TeamCraft	✓	✓	VS	CD	✓	✓	EGA	4+	55,000+	55,000+

pabilities, and new types of visual backgrounds. To evaluate existing techniques using our benchmark, we have designed several baseline models to work within the framework and compare their performance. Our results highlight that current approaches to vision-conditioned collaboration and task planning encounter significant challenges when tested within *TeamCraft*'s complex and dynamic environment, especially when it comes to generalizations.

In summary, the main contributions of this paper are:

1. *TeamCraft*, a new multi-modal multi-agent benchmark with its associated large-scale dataset encompassing complex tasks challenging multi-agent systems in a wide variety of generalization scenarios.
2. Extensive experiments and analyses on state-of-the-art multi-modal multi-agent models, uncovering their strengths and weaknesses to inform and inspire future research.
3. To ensure reproducibility and encourage future works from the community, we open source the entire platform, its training and evaluation code, and release the model checkpoints and training data at

<https://github.com/teamcraft-bench/teamcraft>

2. Related work

2.1. Platforms for Multi-Agent Systems

The recent success of multi-agent reinforcement learning (MARL) methods [32–34, 61] has attracted attention, as these methods explore cooperation and competence behaviors among agents. However, many of the methods are evaluated in simplified 2D environments [5, 26, 39, 52, 55]. Recent work on embodied multi-agent benchmarks has considered more realistic tasks and environments [6, 15, 29, 30, 42], but it often relies on certain privileged sensor information of

the environment [44, 45, 65]. Additionally, subject to environmental constraints, these works often have limited set of tasks [22, 54] related to navigation and simple interactions such as object rearrangement [53]. By comparison, *TeamCraft* is based on Minecraft, a three-dimensional, visually rich open-world realm characterized by procedurally generated landscapes and versatile game mechanics supporting an extensive spectrum of object interactions, providing rich activities ripe for intricate collaborations.

2.2. Embodied Language-Guided Benchmarks

Several researchers have looked at the problem of using natural language as the interface between embodied agents, either in the form of task specifications [16, 47, 48, 67], question answering [8, 18, 35, 36], instruction following [2, 12, 13, 23, 40, 41, 56], or as means of task coordination [27, 37]. VIMA-Bench [24] builds on previous efforts in language-guided robotic manipulation [38, 46, 63] and uses multi-modal prompts as uniform task specifications for object manipulation. *TeamCraft* extends multi-modal prompts to the multi-agent domain and uses them to specify a wide variety of collaborative tasks that require object interaction and navigation.

2.3. Benchmarks Based on Minecraft

Malmo [25] marks the advent of a Gym-style platform tailored to Minecraft games. It paves the way for subsequent single-agent works such as MineRL [19], Voyager [57], and MineDojo [11]. Marlo [43] extends Malmo to multi-agent scenarios, but the small number of task variations limit generalizations. Similar to our work, MindAgent [17] and VillagerBench [10] focus on multi-agent collaboration in a multi-task setting. However, both of these use purely state-based observations, while *TeamCraft* tackles the more

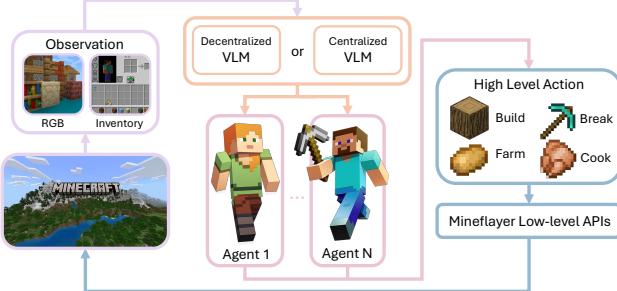


Figure 1. The *TeamCraft* platform consists of three main components: (1) a Minecraft server that hosts the game as an online platform, (2) Mineflayer, which serves as the interface for controlling agents in the server, and (3) a Gym-like environment that provides RGB and inventory observations to the models, allowing control of multiple agents through high-level actions.

challenging problem of learning to collaborate from multi-modal perceptions. Table 1 compares *TeamCraft* with prior benchmarks.

3. TeamCraft Benchmark

3.1. Problem Formulation

Assume that an embodied multi-agent system comprised of N agents needs to complete a complex task involving navigation and object manipulation. The task is specified in a multi-modal prompt $x_L = \{x_l\}_{l=1}^L$, which is a sequence of interleaved language and image tokens with length L . At time step t , each agent receives partial observation $o_n^t \in O$ from the full observation space O . To complete the task, each agent can choose to perform a high level action $a_t \in A$ from the full set of action A . The action can be further decomposed into a sequence of low level control signals.

3.2. Simulation Environment

TeamCraft utilizes Minecraft as its foundational simulation environment, offering a complex, open-world setting for multi-agent interactions. With a Gym-like environment, it facilitates the execution of intricate multi-agent commands via self-explanatory skills. Figure 1 illustrates the platform architecture. High level skills from the model can be translated into low level control signals via nested API calls through Mineflayer¹. After execution, visual observation of each agent are rendered and provided as input to the model.

Multi-Modal Prompts: In our work, the multi-modal prompt x_L consisting of a language instruction interleaved with a set of orthographic projection images (i.e., top, left, and front views) for task specification. Depending on the specific task, the images can specify either the initial states, intermediate states or the goal states.

¹<https://github.com/PrismarineJS/mineflayer>

Observation and Actions: To mimic real world settings of embodied visual agent teaming, we use first-person view RGB image and inventory information as the observation o_n . The action space A involves high-level self-explanatory skills such as *obtainBlock* to obtain a block and *farmWork* to farm a crop. Most actions take three input parameters, including (1) agent name such as *bot1*, as the action-executing entity, (2) item name such as *dirt*, and (3) a 3D vector indicating the position of the target. There are 8 types of actions in total. A complete list of actions are described in Appendix A.

3.3. Task Design

TeamCraft introduces a variety of complex and interactive tasks that challenge the agents’ capabilities in planning, coordination, and execution within a collaborative and dynamic environment. Each task is designed to test different facets of MA interaction, including role distribution, real-time decision-making, and adaptability to changing environments. Task examples are shown in Figure 2 and the corresponding prompt examples are shown in Figure 3.

Building: Agents erect a structure based on a provided orthographic view blueprint. Each agent possesses a unique inventory of building blocks necessary for the construction. Successful completion requires visual cognition to associate blueprint components with inventory items, spatial reasoning to reconstruct a 3D structure from 2D images and map it to 3D coordinates for action targets, and collaborative coordination with other agents to resolve action dependencies. For example, an agent cannot place a floating block and should wait for another agent to build the supporting block first.

Clearing: Agents are required to remove all blocks from a specified area. Besides spatial understanding and awareness of action dependencies, agents must employ appropriate tools to break blocks, which vary in durability, thereby requiring multiple interactions for complete removal. The use of correct tools can dramatically reduce the time required to remove blocks. Thus agents must coordinate task assignments to optimize block-breaking efficiency. Strategic coordination is essential in this task as agents need to dynamically decide which blocks to target based on their current tools, and assist each other even without the optimal tools when necessary.

Farming: Agents sow and harvest crops on designated farmland plots. They must monitor crop growth stages, from newly planted to fully grown, and harvest only when crops reach maturity. Efficient task completion requires spatial reasoning to select appropriate farmland, visual cognition to assess crop maturity, and continuous updating of farmland states based on other agents’ actions. As the available farmland exceeds what is needed, understanding other agents’ actions to avoid redundancy, and dynamically allocating sub-tasks based on positions, available seeds, and crop maturity

	Building	Clearing	Farming	Farming	Smelting	Smelting
Scenes	village	snow_mountain	village	swamp	ice_on_water	desert_village
Base	cyan_concrete	gold_block	hay_block	obsidian	oak_wood	glass
Goal	Build 1x2x4 building	Clean 3D building	Potato *3	wheat *4	cooked_mutton *1	smooth_quartz *2
Object	[dirt, wool, fence, sandstone, sponge]	[grass_block, dirt, birch_log, bookshelf,]	-	-	[birch_planks, sheep]	[oak_planks, quartz_block]
Agent	3	3	2	2	3	2
Inventory	[dirt, wool, fence, sponge, log, stone]	[stone_axe, stone_sword]	[carrot, beetroot]	[wheat_seeds, carrot, potato]	[iron_pickaxe, iron_axe, iron_sword]	[iron_pickaxe, iron_axe]
Demonstration						

Figure 2. We present example task configurations, as a combination of distinct biomes, playground base blocks, task goals, target blocks materials and agent counts. Agents are initialized with unique inventories, which provide them with different capabilities to complete various activities. A detailed distribution is provided in [Appendix J](#).

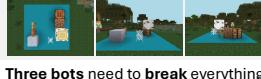
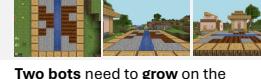
	Building	Clearing	Farming	Smelting
System Prompt	Three Orthographic Views			
	Language Instruction	Three bots need to build a building on the platform. Write actions for bot1, bot2, bot3 based on given observation.	Three bots need to break everything on the platform. Write actions for bot1, bot2, bot3 based on given observation.	Two bots need to grow on the platform. The goal is to get 4 carrot. Write actions for bot1, bot2 based on given observation.
Observation	First Person View			
	Inventory Information	bot1 has 5 bricks, 3 iron_ore, bot2 has 2 sea_lantern, bot3 has 1 brick...	bot1 has a stone_axe, bot2 has a stone_pickaxe, bot3 has a stone_sword...	bot1 has 3 carrot, 1 potato, bot2 has 3 carrot, 2 beetroot...
Action		placeItem(bot1, 'bricks', (-1,0,-1)) placeItem(bot2, 'oak_planks', (0,0,0)) placeItem(bot3, 'iron_ore', (0,0,-1))	mineBlock(bot1, (-1,0,1)) mineBlock(bot2, (-2,0,0)) mineBlock(bot3, (-1,1,1))	farmWork(bot1, (1,-1,1), sow, 'carrot') farmWork(bot2, (-1,-1,-2), sow, 'carrot') putItem(bot1, 'sandstone', (0,0,-1)), obtainBlock(bot2, (2,0,0)), obtainBlock(bot3, (1,0,-3))

Figure 3. Multi-modal prompts are provided for all tasks. The system prompt includes both the three orthographic views and specific language instructions. Observations consist of first-person views from different agents, along with agent-specific information.

are essential. For example, some agents can sow while others are harvesting, stop when the total yield meets the goal.

Smelting: Agents obtain processed items using furnaces by gathering materials and coordinating actions. They collect resources from the environment, by harvesting blocks or killing mobs, or use existing inventory items to produce goal items like cooked food or refined materials. Agents also need to gather fuel before they can make use of furnaces. Efficient task completion requires spatial understanding to locate furnaces and resources, and coordinating actions with inter-agent dependencies. For instance, if one agent is collecting beef, others should focus on gathering fuel rather than duplicating efforts. Working as a team to use limited furnaces efficiently is crucial, rather than each agent independently smelting their own goal item.

3.4. Centralized and Decentralized Agents

TeamCraft supports centralized and decentralized control.

Centralized Agents: The centralized model is given the observational data of all agents, including the first-person

view, action history, and inventory information. Based on these comprehensive data, the model generates the actions for all agents simultaneously. This approach leverages the full scope of information available in the environment to coordinate and optimize the actions of all agents collectively.

Decentralized Agents: The decentralized models do not receive information about other agents except for the initial inventory of the team. Each model generates actions solely for the individual agent based on its limited view. This setting simulates a more realistic scenario where agents operate independently with restricted information, focusing on their actions absent of any centralized coordination.

3.5. Diversity

The tasks are complex and challenging, testing multi-agent systems in diverse settings. [Table 2](#) provides task statistics and variants, with visual diversity detailed in [Appendix C](#).

Object Diversity: More than 30 target object or resource are used in tasks. Objects, such as a fence, an anvil, or a stone block, have different shapes and textures. Farm crops

have different visual appearances during growth stage. The smelting task has resources with different appearances, such as chickens or rabbits.

Inventory Diversity: Agent’s inventory might include essential items mixed with non-essential ones (i.e., distractors), realistically simulating scenarios where agents must choose the right materials for specific tasks while managing inventory constraints. Agents are also provided with random tools for each task. Having the appropriate tools significantly enhances efficiency in tasks like clearing. For smelting, some resources must be collect by agent with specific tools.

Scene Diversity: More than 10 scenes are included in the tasks, covering biomes such as village, mountain, forest, swamp, desert, etc. Tasks take place on grounds with diverse textured bases such as glass, concrete, and quartz. Certain tasks may involve additional complexity, including farmland which are intermixed with non-plantable blocks.

Goal Diversity: Each task requires achieving a varying number of goal targets. Building requires different blocks placed into various shapes, categorized based on dimensionalities, e.g., 2D (all blocks are at the same level) or 3D (some blocks are on top of others). Farming requires various target crops and yields. For the smelting task, the target object is sampled from various food or processed items.

3.6. Tasks and Expert Demonstrations Generation

To create a rich learning environment and effective imitation learning dataset, systematic scenario design and data collection methods are employed, as follows:

Task Generation: Variables from a diversity pool, such as agent counts, scenes, and goals, are sampled to establish task configurations. Specifically, a solvable task is formulated by rejection sampling of the essential task variables. "Solvable" implies that the task can be completed within the Minecraft world rules and is within the agents' capabilities. For example, in smelting tasks, fuel must either be available to collect in the scene or directly accessible in the inventory.

Planner-Based Demonstrations Generation: Given the task specifications, a planner assigns actions to agents at every time step, utilizing privileged information of the environment. Assume agent i performing action j , the planner optimizes a cost function designed to minimize total task completion time T , idle actions E_i , action dependencies D , redundant actions U , and the cost c_{ij} for agent i performing action j :

$$C = w_1 T + w_2 \sum_{i=1}^N E_i + w_3 D + w_4 \sum_{i=1}^N \sum_{j \in A_i} c_{ij} + w_5 U \quad (1)$$

where w_1, w_2, w_3, w_4 , and w_5 are weighting coefficients. Details of the weights are available in [Appendix D](#).

Table 2. Task variants and dataset statistics

	Building	Clearing	Farming	Smelting
# Action Sequences	2 – 6	2 – 9	2 – 7	2 – 8
# Agents	2 – 3	2 – 3	2 – 3	2 – 3
# Tools	–	1 – 4	–	1 – 4
# Scenes	6	5	4	5
# Base Types	10	11	9	11
# Furnaces	–	–	–	1 – 2
# Target Block Types	19	16	3	13
# Target Block Counts	5 – 12	4 – 9	2 – 14	1 – 4
# Fuel Types	–	–	–	12
# Resource Types	–	–	–	20
# Dimensional Shapes	2	2	2	1
# Placement Shapes	7715	12724	13188	8885
# Total Demonstrations	14998	14641	14815	10803
# Test Set	50	50	50	50
# Generalization Set	200	200	150	200
# Generalization Conditions	4	4	3	4

As shown in [Table 2](#), we generated 55,000 unique task variants, each with one demonstration. A demonstration consists of a multi-modal prompt as task specification, including three orthographic view images representing task initial states or goal states and the corresponding language instructions. At each time step, agent inventories, first-person RGB observations and actions are recorded.

3.7. Test Set and Generalization Set

TeamCraft features a test set, where agents are initialized with random position, orientation, and inventory. Other variables follow the same distribution as training. To evaluate the model generalization, we further designed a generalization set with hold-out elements excluded from training data. In general, we withheld test cases involving *four agents*, whereas the training data include only two or three agents. We also introduced unseen *scenes* not present during training. In addition to these general hold-outs, we implemented task-specific exclusions as following: 1) Building: novel *shapes* and *materials* to build. We exclude 8 block placement *shapes*, defining how target blocks are arranged on the ground. These shapes varied in complexity, containing 5 to 12 blocks in both 2D and 3D configurations. Additionally, we omitted 3 block *materials* appeared in clearing but not in building. 2) Clearing: novel *shapes* and *materials* to clear. We held out 6 block placement *shapes* with block counts ranging from 4 to 9. We also excluded 3 block *materials* present in building but absent in clearing. 3) Farming: novel *crops* to farm and collect. 4) Smelting: novel number of *furnaces* and *goal* objects. We excluded 4 unseen *goal* objects and introduced scenarios with novel number of *furnaces* in the scene. As shown in [Table 2](#), with 50 samples per task for the test set and each generalization condition, our benchmark contains a total of 950 test cases.

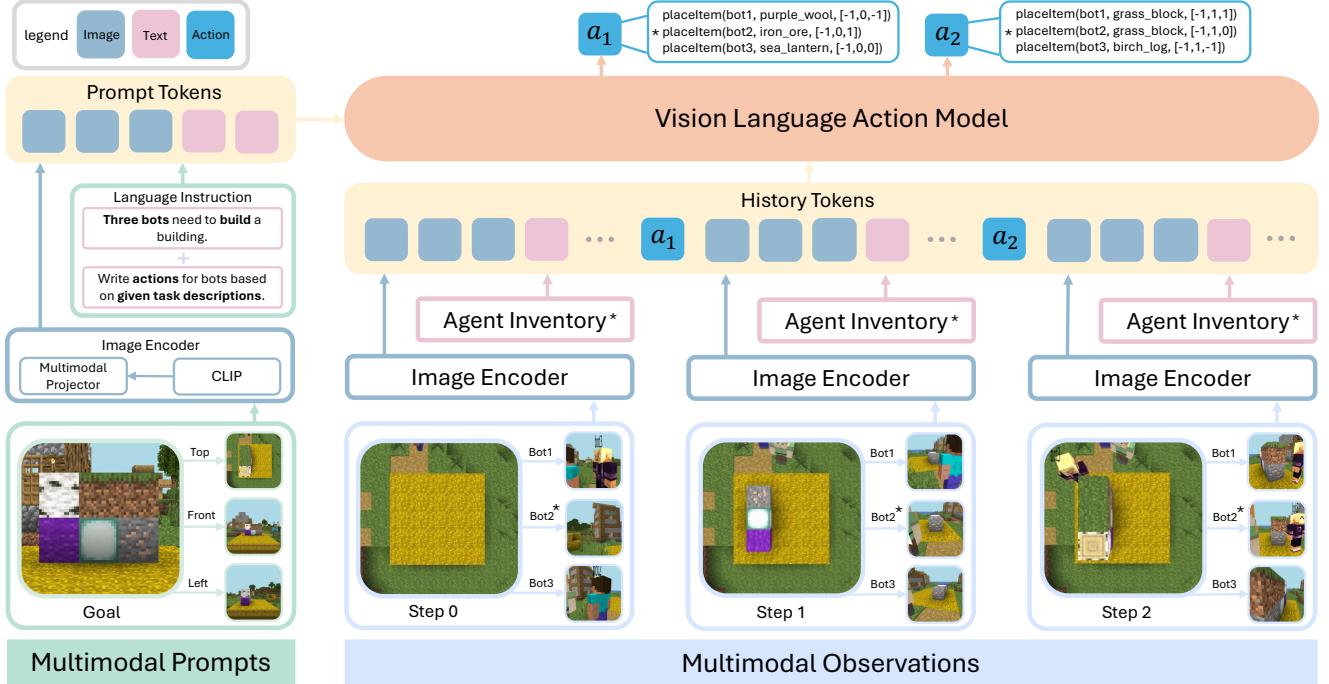


Figure 4. The architecture of the TeamCraft-VLA model. Multi-modal task specifications combining three orthographic views images of the task goal states and corresponding language instructions are encoded as initial input to the model. Agents inventories and visual observations are further encoded in each step to generate actions for agents. For decentralized setting, the model only has access to one agent’s information, exemplified by Bot2: items associated with a * represent the fact that only the data associated with agent 2 are available.

4. Experiments

4.1. Baselines and Ablations

TeamCraft-VLA: We introduce TeamCraft-VLA (Vision-Language-Action), a multi-modal vision-language action model designed for multi-agent collaborations. As shown in Figure 4, the model first encodes multi-modal prompts specifying the task, then encodes the visual observations and inventory information from agents during each time step to generate actions. Following [28], the VLA model architecture consisting of a CLIP encoder for images, a projector to align the image features with language model. We use CLIP ViT-L/14 as the visual encoder and a linear projector for modality alignment. The model is trained on the demonstration data for three epochs before convergence.

Proprietary VLA: We use GPT-4o as the proprietary VLA. Specifically, we use similar prompt structures as the centralized finetuned TeamCraft-VLA model, with additional task information in the initial system prompt to provide background knowledge of the task. The system prompt contains recipes, input, output formats, all available blocks, items, workspace limitations, and one successful rollout of a similar task in the same task family. At the first step, we additionally provide the first user prompt, where the model is given a specific multi-modal task specification accompanied

by initial visual observations and inventory details of the agents. Based on the system prompts and user prompts, the model predicts the actions. As the interaction progresses with subsequent prompts, the context is maintained and expanded with the addition of prior responses and updated visual data. Detailed prompts are available in Appendix H.

Grid-World Settings: In order to understand the impact of learning in multi-modal environment as opposed to purely text-based or state-based environment, we perform an ablation study by translating the *TeamCraft* environments into a 3D grid-world. We retain the same prompt structure of the training data used in the TeamCraft-VLA models, with the main difference being that environmental information (i.e. visual observations and three orthographic view images) is now represented in text, describing the voxel coordinate of each block, e.g. "brick is at (2,3,0), stone is at (2,3,1)...". We fine-tuned a LLM in centralized setting with variance in the dataset size (10%, 50%, and 100% of the total data) for three epochs before convergence.

Ablations: We performed a total of 15 ablation studies, varying in dataset sizes (10%, 50%, and 100% of the total data), control settings (centralized and decentralized), experiment settings (Multi-modal and Grid-World) and sizes of the VLA model (7B and 13B).

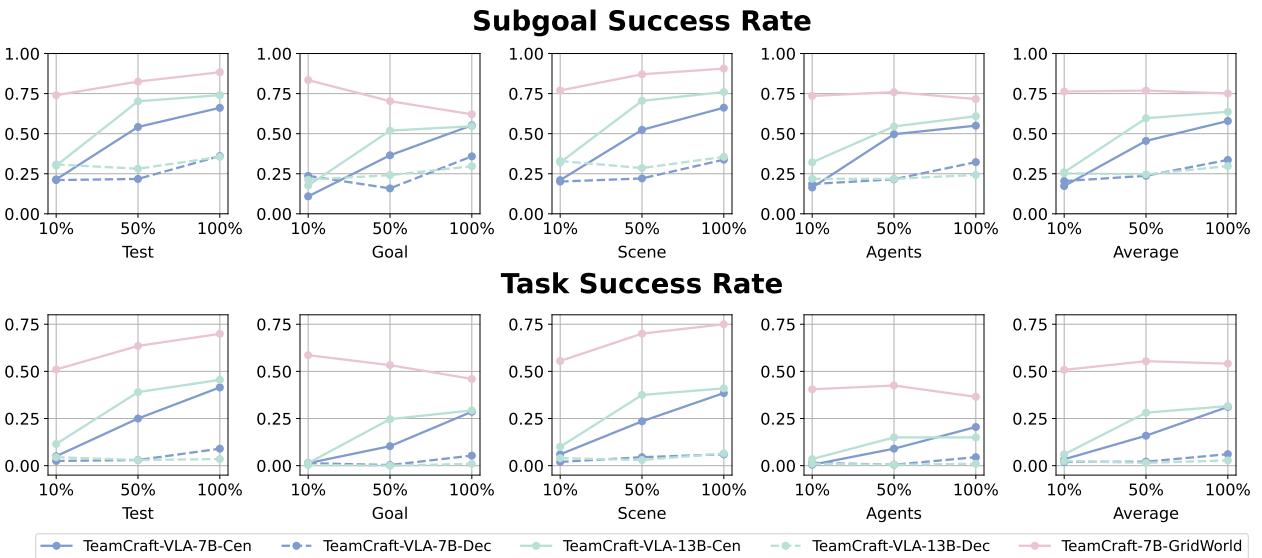


Figure 5. Subgoal success rate and task success rate across centralized, decentralized and grid-world settings. The leftmost column displays the *Test* category, which shares similar data distribution as training. The *Goal*, *Scene* and *Agents* categories represent generalization tasks involving unseen goals, scenes, and tasks involving four agents, respectively. Average performance is presented in the rightmost column.

4.2. Evaluation Metrics

We evaluated the performance of the methods based on three key metrics: task success rate, subgoal success rate and redundancy rate.

Subgoal Success Rate: This metric evaluates the effectiveness of agents in completing tasks. Given M test cases, each test case m has s_m^g subgoals, and agents complete s_m^d subgoals. The subgoal success rate SGS is defined as

$$SGS = \frac{1}{M} \sum_{m=1}^M \frac{s_m^d}{s_m^g} \quad (2)$$

Specifically, subgoals are designed based on the task requirements, i.e. the number of blocks to be built for building and the number of target objects to be created for smelting.

Task Success Rate: This metric indicates the proportion of test cases that the model can successfully complete from start to finish. Specifically, the task success rate TS is defined as:

$$TS = \frac{1}{M} \sum_{m=1}^M \mathbb{1}[s_m^d = s_m^g] \quad (3)$$

A higher success rate reflects the model’s ability to consistently achieve the desired outcomes in various scenarios.

Redundancy Rate: This metric assesses whether multiple agents are performing the same action at the same time, which would lead to conflicts. Assume p_m is the total number of actions for test case m and q_m the number of conflicts

between agents, the redundancy rate RR is defined as:

$$RR = \frac{1}{M} \sum_{m=1}^M \frac{q_m}{p_m} \quad (4)$$

A lower redundancy rate indicates better task allocation among agents and a higher level of cooperative efficiency.

4.3. Evaluation Results

We evaluated the subgoal success rate and task success rate of the models. As illustrated in Figure 5, our analysis and findings are discussed below:

Success Rate: For both the 7B and 13B models, the subgoal success rate and task success rate fall short of optimal performance. This is particularly evident in challenging tasks such as smelting, with both subgoal and task success rates below 40%. This highlights the inherent difficulty of the designed tasks and underscores the current limitations of VLA models in handling multi-step, sequentially dependent processes.

Across Model Size: In Figure 5, we observe that as training data increased, the performance of the 7B model approaches that of the 13B model, especially when generalizing to novel goals and number of agents. This suggests that scaling up model sizes blindly do not guarantee success.

Multi-Modal Environment vs. Grid-World: The performance of the language model in the text-based Grid-World significantly surpasses VLA models in multi-modal settings. This suggests that state descriptions provided purely in text

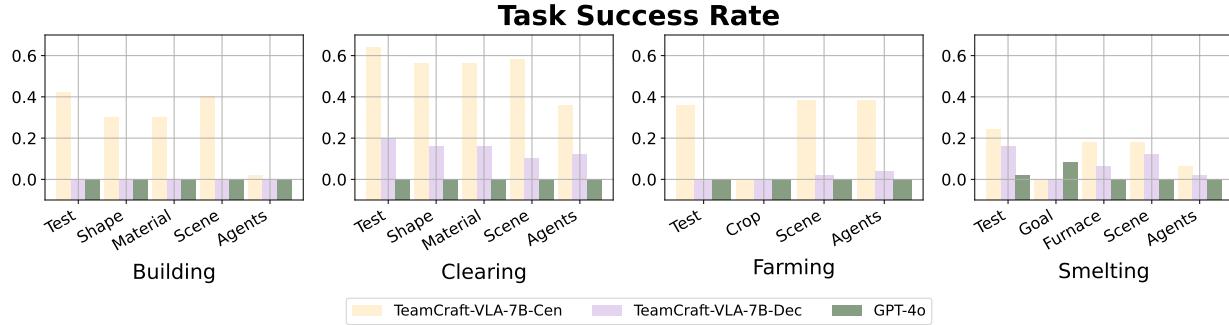


Figure 6. Task success rates of centralized and decentralized VLA model and GPT-4o. Models are trained with the full data except GPT-4o. TeamCraft-VLA-7B-Cen outperforms the other two methods by a significant margin across nearly all variants.

format are less challenging for models than multi-modal inputs, underscoring a notable gap in current VLA models' ability to effectively interpret visual information. For the language model, we observe a surprising trend in the *Goal* and *Agents* splits: training with more data lower the success rate. This decline suggests that the generalization capacity for certain task categories actually diminishes as training goes on. One possible cause is when exposed to more data, the model relies more heavily on patterns specific to the training examples, limiting its ability to adapt to unseen scenarios.

Generalization Splits: For VLA models, performance generally drops when models transfer to novel generalization splits, especially in the *Goal* and *Agents* categories. The *Scene* split primarily tests image understanding, while the *Goal* and *Agents* splits emphasize task planning and allocation, critical factors in multi-agent systems. This indicates that VLA models still struggle with planning for unseen goals and adapting to variable numbers of agents.

Scaling Law: As training data increases, we observe significant improvements in both subgoal and task success rates across centralized and decentralized settings, underscoring the importance of dataset size for achieving better performance. The improvement is particularly pronounced when the training data increases from 10% to 50% in centralized settings. This suggests that while more data generally leads to better performance, gains diminish beyond a certain point, especially in the decentralized setting.

Centralized vs. Decentralized: Figure 6 compares centralized and decentralized settings in terms of subgoal and task success rates across all task variants. Centralized tasks exhibit significantly better performance across nearly all variants, highlighting the challenge of effective planning with partial information. This finding also demonstrates that multi-agent systems cannot be simplistically modeled as single agents interacting with environments containing other agents. In decentralized settings, the absence of comprehensive agent modeling is particularly impactful, especially for

Table 3. Comparison of TeamCraft-VLA model redundancy rates.

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	0.01	0.02	0.01	0.01	0.01
TeamCraft-VLA-7B-Dec	0.13	0.12	0.13	0.24	0.15

cooperation-intensive tasks like "Farming" or "Building".

Redundancy Rate: Table 3 compares redundancy rates between centralized and decentralized settings. Our results show that decentralized agents exhibit significantly higher redundancy rates than centralized agents, indicating reduced efficiency in task planning and allocation. This inefficiency becomes even more pronounced as the number of agents increases, creating greater challenges for effective task allocation. In decentralized settings, the absence of centralized control complicates the avoidance of redundant work, as each agent must independently infer the intentions of others to prevent duplication. By contrast, a centralized controller can efficiently assign distinct tasks to each agent, minimizing overlap and enhancing overall efficiency. These findings suggest that VLA models lacking explicit mechanisms to understand or infer the actions of other agents, highlighting a critical need for improved inter-agent communication and awareness within decentralized systems.

GPT-4o Result: We evaluate GPT-4o on in a one-shot prompt learning setup and it failed on almost all test cases. A detailed analysis reveals that GPT-4o struggles with mapping block coordinates based on visual inputs, demonstrate a lack of 3D spatial reasoning needed for accurate task execution. This shortcoming severely impacts performance, since most of our tasks require precise spatial orientation and alignment. For example, in building task, a brick should be placed at (8,0,8), while the output of model is "placeItem(bot1, 'bricks', (7,0,9))" which leads to wrong execution.

4.4. Qualitative Analysis

We performed a qualitative analysis across three generalization splits, examining how models handle novel goals, new scenes, and novel number of agent:

Goals: When faced with novel goals, the models struggle to generalize beyond familiar items and often fail to adapt to specific, unseen objectives. For example, in the "farming" task, if instructed to farm beetroot—a crop not encountered in training—the model might generate a command like "farm_work(bot1, (9,3,3), 'sow', 'beef')," causing Bot1 to sow "beef", which appears in the training data for "smelting". This behavior reflects the model's reliance on similar, previously seen items in the training data and reveals its limited ability to infer new tasks based solely on partial similarity.

Object State Recognition: VLA models show strong generalization to new scenes, performing comparably to the *Test* set. However, errors often arise in recognizing object states. For example, in "farming" tasks, agents may harvest crops before they are fully grown due to challenges in identifying crop states, especially when encountering new scenes. This highlights limitations in precise object state recognition when operating within unseen environments.

Agents: For generalization to four agents, models frequently ignore the fourth agent and assigning tasks inefficiently only to two or three agents. For example, for the building task, the model predicts the action sequences {"placeItem(bot1, 'birch_log', (4,4,7))", "placeItem(bot2, 'sandstone', (4,4,6))", "placeItem(bot3, 'dirt', (3,4,6))"} with the fourth agent overlooked, reducing productivity and sometimes preventing timely task completion. This limitation becomes especially evident in tasks requiring full coordination, such as "Building." In these tasks, each of the four agents holds unique blocks in their inventory, and all agents must contribute their specific block to a shared platform to complete the structure. The model's inability to distribute tasks effectively across all agents often leads to incomplete struc-

tures or outright task failure. This highlights a significant limitation in precise coordination and workload distribution necessary for successful multi-agent collaboration.

5. Conclusions

We have presented *TeamCraft*, a benchmark for multi-modal multi-agent collaborative task planning in Minecraft. The benchmark consists of challenging collaborative tasks and evaluation splits designed to systematically test multi-modal agents across novel goal configurations, unseen numbers of agents, and unseen scenes. We conducted extensive experiments and analyses to pinpoint the limitations of the current models and identified promising research directions for collaborative multi-modal agents.

5.1. Limitations and Future Work

1. Given the limited capacity of existing multi-agent VLA models, *TeamCraft* relies on MineFlayer as an oracle controller to execute skills predicted by the models. Enabling VLA models to directly control multiple agents via low-level control [58, 59] would be important future research.
2. We have trained the models using procedurally generated multi-agent demonstration data. Learning from noisy but more diverse real-world demonstrations of human players can potentially further strengthen model generalization [3, 11].
3. Currently decentralized *TeamCraft* agents rely solely on implicit communication [22]; i.e., through passively perceiving other agents and the environment, to gather information and to collaborate. Enabling agents to communicate explicitly via natural language [23, 37, 40] has great potential in avoiding redundant actions and increasing efficiency.
4. Multi-player video games have been widely used as testbeds for human-AI collaboration [1, 5, 14]. Extending *TeamCraft* with human players is a promising research direction.

Appendices

A. High Level Skills

The action space of agents mainly involves high-level self-explanatory skills such as *obtainBlock* and *farmWork*. We provided 8 such skills. Most skills take three input parameters, including 1) agent name such as *bot1*, as the action executing entity, 2) item name such as *dirt*, which strongly associated with task goal or agent's inventory, 3) a vector indicating the position of the target on the test field.

For example, `obtainBlock(bot1, new Vec3(1, 0, 1))` takes the agent name *bot1* and a 3D vector $(1, 0, 1)$ as its arguments. It directs *bot1* to perform multiple actions in Minecraft via APIs provided by Mineflayer. First, it controls *bot1* to *goto* a diggable position for block $(1, 0, 1)$, then has *bot1*'s vision ray cast to the block at $(1, 0, 1)$ using the *lookAt* action. Next, it commands *bot1* to *equip* a proper tool that can dig the block at $(1, 0, 1)$ most efficiently, and then instructs *bot1* to dig the target block. Once the target block has been mined, *bot1* will *goto* the position where the block item dropped and collect it.

Similarly, `farmWork(bot2, "sow", "potato", new Vec3(2, 0, 4))` takes the agent name *bot2*, action type "sow" (as opposed to "harvest"), crop seed item "potato", and a 3D vector $(2, 0, 4)$ as its arguments. It directs *bot2* to *goto* a placeable position for farmland at $(2, 0, 4)$, then check if the seed is a valid item—that is, a crop seed available within *bot2*'s inventory. It then checks if the farmland at $(2, 0, 4)$ is plantable. Finally, it instructs *bot2* to *lookAt* the farmland and *sow* it with the seed "potato".

Table 4 documents all the skills, which are implemented in JavaScript code with Mineflayer APIs.

B. Low Level Atomic Actions

High level skills are processed through multiple stages before reaching the final execution APIs. At each time step, *TeamCraft* accepts a list of skills as input, with a maximum length equal to the number of agents involved in the current task and a minimum length of zero. Each agent can perform at most one skill per time step. The updated list of skills is then passed into the JavaScript environment along with the predefined atomic actions. Each atomic action is processed simultaneously, meaning that agents' actions are executed concurrently rather than sequentially. This avoid the dependency issue that might occur in sequential execution. For example, if one agent's action is executed ahead of another's, the first agent may block the location where the next agent intends to place a block. The agent whose atomic action is executed first will have a higher chance of success, potentially altering the dynamics of the multi-agent setting. Executing actions concurrently ensures fairness among agents and

maintains the equivalence of the multi-agent environment.

C. Visual Diversity

TeamCraft uses a set of visual variate to provide a visual rich environment. Each task is constructed from a random number of agents, in a randomly selected scene, achieving different goal on playground built by different base block.

C.1. Shared Elements

Each task begins with a basic setting involving multiple agents on a playground. Each agent has a unique skin, as illustrated in Figure 23, and is rendered as a two-block-high character. The playground combines a base platform spawned within a Minecraft biome. The base block is also randomly selected from a pool, shown in Figure 23. Each biome offers variations in special surrounding blocks, designs, and environments.

For example, the seaside village biome is a village near the sea with houses made of oak wood and cobblestone, decorated with flowers and cow sheds, as shown in Figure 28. It also features a nearby farm surrounded by oak logs (Figure 29). Another variation of village is the desert village biome, built from acacia planks, acacia logs, and sandstone, blending seamlessly with the desert's arid terrain, shown in Figure 30. Figure 31 illustrates a biome that is located on half of the mountains, where a small flat land protruding from a cliff. Additional examples of biomes used are shown in Figure 32, Figure 33, and Figure 34.

C.2. Task Specific Diversity

Clearing task uses a random set of blocks as its targets, illustrated in Figure 24. **Building** task also uses a random set of blocks as its target, with some blocks shared with clearing task, as illustrated in Figure 25. Unlike other tasks, the **Farming** task does not use a regular base. The playground is constructed from a combination of farmland for planting crops, water blocks, and randomly selected unfarmable blockers from the base that replace some of the farmland. An example is shown in Figure 39. Each corps used in farming task has its own grown stage with different appearances, shown in Figure 26. **Smelting** task requires a wide varieties of resources to achieve its goal. Resources could be either entity, block, or item, shown in Figure 27.

Detailed statistics of each task are presented in Table 11, Table 12, Table 13, and Table 14 of Appendix J.

D. Planner for Expert Demonstration

TeamCraft employed a planner to assign actions to each agent at every time step, utilizing perfect knowledge of the task including goal object positions, agents' inventories, and each agent's efficiency in performing actions. The planner optimizes actions using a cost function designed to minimize

Table 4. Action space of *TeamCraft*.

Type	Arguments	Description
placeItem	BotID, ItemType, Location	BotID places an item of ItemType at the specified 3D Location.
mineBlock	BotID, Location	BotID mines a block at the specified 3D Location.
farmWork	BotID, Location, Action, ItemType	BotID performs an Action (sow or harvest) on ItemType at the specified 3D Location.
obtainBlock	BotID, Location	BotID obtains a block from the specified 3D Location.
putFuelFurnace	BotID, ItemType, Location	BotID places an ItemType as fuel into a furnace at the specified 3D Location.
putItemFurnace	BotID, ItemType, Location	BotID inserts an ItemType into a furnace at the specified 3D Location.
takeOutFurnace	BotID, ItemType, Location	BotID removes an ItemType from a furnace at the specified 3D Location.
killMob	BotID, Location	BotID engages and eliminates a mob at the specified 3D Location.

the total time to complete the task, reduce idle times for agents, minimize action dependencies to prevent agents from waiting on others, maximize parallelism of actions, assign tasks to the most efficient agents, and eliminate redundant or unnecessary actions. The cost function considers the following components:

Minimize Total Task Completion Time T : Denoted by $\min T$, our primary objective is to reduce the overall time required to complete the task, measured in time steps until the last agent finishes their final action.

Minimize Idle Actions for Each Agent E : Denoted by $\min \sum_{i=1}^N E_i$, we minimize the total empty actions, the sum of empty action E_i preformed by agent i .

Minimize Action Dependencies Across Agents D : Denoted by $\min D$, we minimize dependencies cause agents to wait for others to complete certain actions.

Minimize Redundant or Useless Actions U : Denoted by $\min U$, we minimize the total number of redundant or unnecessary actions performed by all agents.

Maximize Action Efficiency: We assign actions to agents with higher capabilities to minimize the overall cost, $\min \sum_{i=1}^N \sum_{j \in A_i} c_{ij}$, where c_{ij} be the cost (inverse of efficiency) for agent i to perform action j .

We assign each component a weight as follows:

$$C = w_1 T + w_2 \sum_{i=1}^N E_i + w_3 D + w_4 \sum_{i=1}^N \sum_{j \in A_i} c_{ij} + w_5 U \quad (5)$$

where w_1, w_2, w_3, w_4 , and w_5 are weighting coefficients that are adjusted for each task as described below:

Building: In the building task, where dependencies are moderate and parallelization is preferred, we place greater emphasis on minimizing idle actions by setting $w_2 = 1.4$ and assign a weight of 0.9 to the other components. This encourages agents to remain active and reduces idle time, enhancing overall efficiency.

Clearing: In the clearing task, using the correct tools can significantly speed up block removal (up to a threefold increase). Therefore, we assign a higher weight of $w_4 = 1.8$ to maximize action efficiency by assigning tasks to the most capable agents. The other weights are set to 0.8 to maintain overall performance while focusing on efficient tool usage.

Farming: Farming task is not heavily constrained by action dependencies, we assign equal weights of 1 to all components, ensuring a balanced consideration of time minimization, idle actions, action dependencies, action efficiency, and redundancy elimination.

Smelting: In the smelting task, which involves comparatively long and highly dependent action sequences, we prioritize minimizing action dependencies by setting $w_3 = 1.8$. The other weights are assigned a value of 0.8 to support this focus, facilitating smoother coordination among agents and reducing waiting times.

D.1. Example Expert Demonstrations

Figure 35 and Figure 36 show a classic example of the building task, which involves three agents building a 2×3 building on the mountain half. Each of the agents has some of the needed blocks in their inventory to build the building. For every time step after Step 0, each of the three agents build one block from the bottom level to the second level.

Figure 37 shows an example of the clearing task. Two agents are assigned to clean the blocks on a 6×6 platform. Each of them has a stone pickaxe in their inventory, which is the efficient tool for breaking "stone-like" blocks. In this case, they are able to break brick and sandstone in just one time step with pickaxe but require two time steps to break "wood-made" blocks such as bookshelf and crafting table. This resulted in time Step 2 and Step 3 having exactly the same visual observation, shown in Figure 38.

Figure 39 and Figure 40 shows an example of two agents farming on a snow mountain for two extra carrots. In Step 1, agent1 and agent2 both sow the carrots on the open ground.

System Prompt

Three bots need to build a building on the platform. Target building is: Put sea_lantern on [0 ,1 ,0]. Put oak_fence on [-1 ,1 ,0]. Put sponge on [0 ,1 ,-1]. Put emerald_block on [-1 ,1 ,-1]. Put dirt on [0 ,0 ,0]. Put bricks on [-1 ,0 ,0]. Put emerald_block on [0 ,0 ,-1]. Put clay on [-1 ,0 ,-1]. .

User Prompt

bot1 has 4 dirt. bot1 has 3 clay. bot1 has 7 emerald_block. bot1 has 1 oak_fence. bot1 has 3 sponge. bot1 has 1 bricks. bot1 has 3 sea_lantern. bot2 has 4 bricks. bot2 has 2 sponge. bot2 has 6 sea_lantern. bot2 has 2 oak_fence. bot2 has 4 emerald_block. bot2 has 1 dirt. bot2 has 3 clay. bot3 has 6 emerald_block. bot3 has 4 oak_fence. bot3 has 2 dirt. bot3 has 2 sponge. bot3 has 3 clay. bot3 has 2 sea_lantern. bricks is on [-1 ,0 ,0]. dirt is on [0 ,0 ,0]. Write the actions for bot1, bot2 and bot3 based on this given observation.

Figure 7. Prompt example for Building task under the grid-world setting.

System Prompt

Three bots need to break everything on the platform. clay is on [-2 ,0 ,-2]. birch_log is on [-2 ,0 ,0]. dirt is on [-1 ,0 ,-2]. crafting_table is on [-1 ,0 ,1]. anvil is on [-1 ,1 ,1]. anvil is on [0 ,0 ,-2]. iron_ore is on [0 ,0 ,1]. cobweb is on [1 ,0 ,1].

User Prompt

bot1 has 1 stone_pickaxe. bot1 has 1 anvil. bot2 has 1 stone_axe. bot2 has 1 crafting_table. bot3 has 1 stone_pickaxe. bot3 has 1 dirt. clay is on [-2 ,0 ,-2]. birch_log is on [-2 ,0 ,0]. iron_ore is on [0 ,0 ,1]. cobweb is on [1 ,0 ,1]. Write the actions for bot1, bot2 and bot3 based on this given observation.

Figure 8. Prompt example for Clearing task under the grid-world setting.

In Step 2 they saw that the carrots are ready to collect and they both collected one carrot in Step 3 and eventually they collected two carrots.

Figure 41 and Figure 42 shows an example of the smelting task, where two agents need to cook two porkchops. In Step 1, one agent is in charge of adding the fuel to the furnace and the other agent tries to kill the pig to obtain a raw porkchop. Since bot2 already has one porkchop, it only requires one additional porkchop. In Step 2, both agents put the porkchop into the furnace and in Step 3, they have 2 cooked porkcops.

E. Grid-World Settings

Under the grid-world setting, we replace the three orthographic view images and first person view images with text descriptions of the task goal and current environment states, and provide them as input to the model. Here we show one example of the prompt construction in each task.

Building: As shown in Figure 7, the system prompt consists of both task description and the target building coordination of each block. The user prompt consists of the built blocks and the inventories of the agents.

Clearing: As shown in Figure 8, the system prompt consists of both task description and the blocks that appeared on the platform initially. The user prompt consists of the blocks that appeared on the platform at current time step and the inventories of the agents.

Farming: As shown in Figure 9, the system prompt consists of both task description and the blocks in the farmland. The user prompt consists of the blocks in the farmland and crops information at current time step and the inventories of the agents.

Smelting: As shown in Figure 10, the system prompt consists of both task description, instructions to craft different items and the blocks in the field. The user prompt consists of the blocks locations at current time step and the inventories of the agents.

System Prompt

Two bots need to grow on the platform. The goal is to get 5 carrot. farmland is on [-3 ,-1 ,-2] with value of 7. cyan_concrete is on [-3 ,-1 ,-1]. water is on [-3 ,-1 ,0]. cyan_concrete is on [-3 ,-1 ,1]. cyan_concrete is on [-3 ,-1 ,2]. farmland is on [-2 ,-1 ,-2] with value of 7. cyan_concrete is on [-2 ,-1 ,-1]. water is on [-2 ,-1 ,0]. farmland is on [-2 ,-1 ,1] with value of 7. cyan_concrete is on [-2 ,-1 ,2]. cyan_concrete is on [-1 ,-1 ,-2]. cyan_concrete is on [-1 ,-1 ,-1]. water is on [-1 ,-1 ,0]. farmland is on [-1 ,-1 ,1] with value of 7. farmland is on [-1 ,-1 ,2] with value of 7. cyan_concrete is on [0 ,-1 ,-2]. farmland is on [0 ,-1 ,-1] with value of 7. water is on [0 ,-1 ,0]. cyan_concrete is on [0 ,-1 ,1]. cyan_concrete is on [0 ,-1 ,2]. cyan_concrete is on [1 ,-1 ,-2]. cyan_concrete is on [1 ,-1 ,-1]. water is on [1 ,-1 ,0]. farmland is on [1 ,-1 ,1] with value of 7. cyan_concrete is on [1 ,-1 ,2]. cyan_concrete is on [2 ,-1 ,-2]. cyan_concrete is on [2 ,-1 ,-1]. water is on [2 ,-1 ,0]. cyan_concrete is on [2 ,-1 ,1]. farmland is on [2 ,-1 ,2] with value of 7. cyan_concrete is on [3 ,-1 ,-2]. farmland is on [3 ,-1 ,-1] with value of 7. water is on [3 ,-1 ,0]. cyan_concrete is on [3 ,-1 ,1]. farmland is on [3 ,-1 ,2] with value of 7.

User Prompt

bot1 has 5 carrot. bot1 has 2 beetroot. bot1 has 3 potato. bot2 has 2 carrot. bot2 has 2 beetroot. bot2 has 2 wheat_seeds. farmland is on [-3 ,-1 ,-2] with value of 7. cyan_concrete is on [-3 ,-1 ,-1]. water is on [-3 ,-1 ,0]. cyan_concrete is on [-3 ,-1 ,1]. cyan_concrete is on [-3 ,-1 ,2]. farmland is on [-2 ,-1 ,-2] with value of 7. cyan_concrete is on [-2 ,-1 ,-1]. water is on [-2 ,-1 ,0]. farmland is on [-2 ,-1 ,1] with value of 7. cyan_concrete is on [-2 ,-1 ,2]. cyan_concrete is on [-1 ,-1 ,-2]. cyan_concrete is on [-1 ,-1 ,-1]. water is on [-1 ,-1 ,0]. farmland is on [-1 ,-1 ,1] with value of 7. farmland is on [-1 ,-1 ,2] with value of 7. cyan_concrete is on [0 ,-1 ,-2]. farmland is on [0 ,-1 ,-1] with value of 7. water is on [0 ,-1 ,0]. cyan_concrete is on [0 ,-1 ,1]. cyan_concrete is on [0 ,-1 ,2]. cyan_concrete is on [1 ,-1 ,-2]. cyan_concrete is on [1 ,-1 ,-1]. water is on [1 ,-1 ,0]. farmland is on [1 ,-1 ,1] with value of 7. cyan_concrete is on [1 ,-1 ,2]. cyan_concrete is on [2 ,-1 ,-2]. cyan_concrete is on [2 ,-1 ,-1]. water is on [2 ,-1 ,0]. cyan_concrete is on [2 ,-1 ,1]. farmland is on [2 ,-1 ,2] with value of 7. cyan_concrete is on [3 ,-1 ,-2]. farmland is on [3 ,-1 ,-1] with value of 7. water is on [3 ,-1 ,0]. cyan_concrete is on [3 ,-1 ,1]. farmland is on [3 ,-1 ,2] with value of 7. carrots is on [3 ,0 ,-1] with value of 0. carrots is on [3 ,0 ,2] with value of 0. Write the actions for bot1, bot2 based on this given observation.

Figure 9. Prompt example for Farming task under the grid-world setting.

System Prompt

Two bots need to craft 2 stone. here are the instructions: Cooking Food: 1. To cook a 'cooked_beef' ... cobblestone is on [-2 ,0 ,2]. furnace is on [0 ,0 ,1]. spruce_planks is on [2 ,0 ,-3]. cobblestone is on [2 ,0 ,-1].

User Prompt

bot1 has 1 iron_sword. bot1 has 1 iron_shovel. bot1 has 1 iron_pickaxe. bot1 has 1 cobblestone. bot1 has 1 spruce_planks. bot2 has 1 spruce_planks. bot2 has 1 iron_shovel. bot2 has 2 iron_pickaxe. cobblestone is on [-2 ,0 ,2]. furnace is on [0 ,0 ,1]. spruce_planks is on [2 ,0 ,-3]. cobblestone is on [2 ,0 ,-1]. Write the actions for bot1, bot2 based on this given observation.

Figure 10. Prompt example for Smelting task under the grid-world setting.

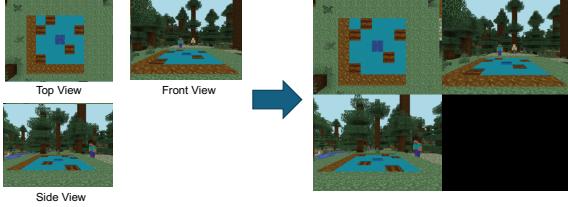


Figure 11. Combining three orthogonal view images into a single composite image as model input.

F. TeamCraft-VLA Implementation Details

We use Vicuna-v1.5 as the LLM backbone. For the visual encoder, we employ CLIP ViT-L/14 to process all input images, including three orthogonal views and the first-person view of the agents. The image embeddings are then projected into the LLM space with a linear projection layer and concatenated with the text embeddings. The combined embeddings are fed into the LLM, which outputs the final action. During training, we froze the visual encoder and projector and only finetune the LLM. All image embeddings are positioned before the text embeddings, separated by "image start" and "image end" tokens. In centralized settings, where the number of images varies depending on the number of agents, we pad a dummy image at the end for training stability if the task involves only two agents. In decentralized settings, the number of image inputs remains unaffected, as the model processes only the first-person view of the current agent, excluding views from others.

We train each model for 3 epochs using the training split, leveraging 8 A100 GPUs with a global batch size of 16. In the centralized setting, training the 7B model takes 36 hours, while the 13B model requires 72 hours. In the decentralized setting, the training duration doubles, with the 7B model requiring 72 hours and the 13B model taking 144 hours. In the grid-world setting, training the 7B model takes 20 hours.

F.1. Arrangement of Three Orthogonal Views

For training and evaluation, we combine the three orthogonal view images into a single composite image by arranging them to the upper-left top-left corner, top-right corner, and the lower-left corner of the composite image. An example of this arrangement is shown in Figure 11. This process is to reduce the number of images provided to the model to conform with the 4096 context length limit.

F.2. Hyperparameters

We present the hyperparameters for VLA training in Table 5.

F.3. Model Output Parsing

The output of the model is a string which will be parsed into the pre-defined high level skills. The string will be first processed by removing special sentence begin token, $\langle s \rangle$,

and ending token $\langle /s \rangle$. It will then be split into a list, where each item is parsed as the skill of one agent.

G. Additional Results of TeamCraft-VLA

G.1. Task Success Rate and Subgoal Success Rate

We show task success rate and subgoal success rate of centralized and decentralized 7B models with different data scales in Table 6, and those of 13B models in Table 7. We compare among different centralized models in Table 8.

G.2. Redundancy Rate

We present a more detailed analysis of redundancy rates, including the 13B models, in Table 9. Both the 7B and 13B models exhibit redundancy issues in decentralized settings. Increasing model size alone does not resolve the redundancy problem in such scenarios.

G.3. Action Sequence Length

We compared the average action lengths across different splits between the 7B and 13B models under both centralized and decentralized settings, as shown in Table 10. In general, decentralized settings require longer action sequences to complete tasks. Among the splits, the *Goal* split is the most challenging, as it demands more actions to accomplish the tasks.

G.4. Case Study

We next present a detailed failure case analysis by categories:

Object Mismatching: As an example (Figure 12), in the farming tasks two agents need to get 10 beetroot. In Step 0, the actions involve a mismatch in the objects; the agents mistakenly sow "beet_seeds" instead of "beetroot_seeds." Consequently, in Step 1, due to the object mismatch, no crops grow on the farmland. As another example, two agents need to get 2 dried kelp in the smelting task (Figure 13). The task requires one bot to put the kelp and the other put the fuel. However, in this example bot1 mistake the object "kelt" to "cobb11".

Task Allocation Failure: This occurs when a task requires four agents. As two examples, four agents must break everything on the platform in the clearing task (Figure 14), and construct on the platform in the building task (Figure 15). Only three agents are assigned distinct actions, leaving the fourth agent idle.

Object State Recognition Failure: As an example (Figure 16), a farming task requires two agents to collect four additional carrots. In Step 0, *bot1* and *bot2* both sow carrots and attempt to harvest them in Step 2. However, at that time, the carrots are still immature and not ready for collection. The mature state of the carrot is shown in Figure 26.

Table 5. Hyperparameters for TeamCraft-VLA

lr	model max length	vision tower	patch size	resolution	language model	optimizer	lr scheduler type	warmup ratio
2e-5	4096	openai-clip-vit-large	14	336*336	Vicuna-v1.5	AdamW	constant_with_warmup	0.03

Table 6. Task success rates and subgoal success rates of the TeamCraft-VLA-7B-Cen and TeamCraft-VLA-7B-Dec models. Subgoal success rates are given in parentheses.

Tasks	Condition	Centralized			Decentralized		
		10%	50%	100%	10%	50%	100%
Building	Test	0.00 (0.12)	0.38 (0.76)	0.42 (0.81)	0.00 (0.18)	0.00 (0.28)	0.00 (0.38)
	Shape	0.00 (0.12)	0.20 (0.67)	0.30 (0.75)	0.00 (0.15)	0.00 (0.25)	0.00 (0.40)
	Material	0.00 (0.13)	0.18 (0.64)	0.30 (0.74)	0.00 (0.13)	0.00 (0.20)	0.00 (0.34)
	Scene	0.00 (0.15)	0.36 (0.73)	0.40 (0.83)	0.00 (0.16)	0.00 (0.21)	0.00 (0.36)
	Agents	0.00 (0.18)	0.02 (0.50)	0.02 (0.57)	0.00 (0.12)	0.00 (0.20)	0.00 (0.14)
Clearing	Test	0.00 (0.13)	0.08 (0.43)	0.64 (0.91)	0.00 (0.45)	0.02 (0.35)	0.20 (0.68)
	Shape	0.00 (0.09)	0.08 (0.34)	0.56 (0.91)	0.00 (0.47)	0.02 (0.27)	0.16 (0.74)
	Material	0.00 (0.10)	0.12 (0.45)	0.56 (0.90)	0.00 (0.48)	0.00 (0.22)	0.16 (0.67)
	Scene	0.00 (0.11)	0.10 (0.44)	0.58 (0.92)	0.00 (0.41)	0.04 (0.37)	0.10 (0.64)
	Agents	0.00 (0.16)	0.14 (0.64)	0.36 (0.81)	0.02 (0.50)	0.02 (0.54)	0.12 (0.60)
Farming	Test	0.14 (0.43)	0.34 (0.60)	0.36 (0.63)	0.02 (0.07)	0.02 (0.14)	0.00 (0.09)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	Scene	0.16 (0.39)	0.34 (0.65)	0.38 (0.67)	0.00 (0.05)	0.00 (0.11)	0.02 (0.07)
	Agents	0.02 (0.18)	0.18 (0.61)	0.38 (0.68)	0.00 (0.08)	0.00 (0.11)	0.04 (0.27)
Smelting	Test	0.06 (0.17)	0.20 (0.36)	0.24 (0.28)	0.08 (0.13)	0.08 (0.09)	0.16 (0.29)
	Goal	0.08 (0.21)	0.04 (0.07)	0.00 (0.00)	0.08 (0.17)	0.00 (0.00)	0.00 (0.00)
	Furnace	0.10 (0.28)	0.10 (0.20)	0.18 (0.20)	0.06 (0.07)	0.06 (0.06)	0.06 (0.16)
	Scene	0.08 (0.19)	0.14 (0.28)	0.18 (0.23)	0.08 (0.19)	0.14 (0.19)	0.12 (0.28)
	Agents	0.00 (0.15)	0.02 (0.24)	0.06 (0.13)	0.04 (0.05)	0.00 (0.02)	0.02 (0.28)

Table 7. Task success rates and subgoal success rates of the TeamCraft-VLA-13B-Cen and TeamCraft-VLA-13B-Dec models. Subgoal success rates are given in parentheses.

Tasks	Condition	Centralized			Decentralized		
		10%	50%	100%	10%	50%	100%
Building	Test	0.00 (0.18)	0.46 (0.80)	0.48 (0.79)	0.00 (0.13)	0.00 (0.18)	0.00 (0.31)
	Shape	0.00 (0.16)	0.30 (0.73)	0.26 (0.69)	0.00 (0.15)	0.00 (0.15)	0.00 (0.32)
	Material	0.00 (0.15)	0.24 (0.65)	0.08 (0.63)	0.00 (0.14)	0.00 (0.14)	0.00 (0.31)
	Scene	0.00 (0.16)	0.38 (0.75)	0.48 (0.83)	0.00 (0.17)	0.00 (0.17)	0.00 (0.28)
	Agents	0.00 (0.16)	0.00 (0.49)	0.04 (0.59)	0.00 (0.14)	0.00 (0.16)	0.00 (0.23)
Clearing	Test	0.04 (0.37)	0.42 (0.83)	0.64 (0.94)	0.00 (0.46)	0.02 (0.62)	0.02 (0.60)
	Shape	0.00 (0.26)	0.42 (0.85)	0.78 (0.96)	0.00 (0.47)	0.00 (0.57)	0.04 (0.58)
	Material	0.04 (0.36)	0.36 (0.83)	0.56 (0.92)	0.02 (0.53)	0.00 (0.60)	0.02 (0.58)
	Scene	0.06 (0.35)	0.44 (0.88)	0.48 (0.90)	0.00 (0.55)	0.02 (0.59)	0.08 (0.64)
	Agents	0.02 (0.55)	0.16 (0.65)	0.16 (0.77)	0.02 (0.50)	0.02 (0.52)	0.02 (0.50)
Farming	Test	0.4 (0.72)	0.62 (0.79)	0.46 (0.73)	0.08 (0.39)	0.04 (0.23)	0.02 (0.33)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	Scene	0.30 (0.69)	0.52 (0.76)	0.44 (0.75)	0.04 (0.32)	0.06 (0.29)	0.10 (0.33)
	Agents	0.12 (0.54)	0.44 (0.79)	0.36 (0.72)	0.02 (0.22)	0.00 (0.19)	0.02 (0.23)
Smelting	Test	0.06 (0.08)	0.22 (0.44)	0.32 (0.59)	0.10 (0.25)	0.06 (0.09)	0.10 (0.19)
	Goal	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.10)	0.00 (0.00)	0.00 (0.00)
	Furnace	0.06 (0.08)	0.20 (0.40)	0.18 (0.38)	0.06 (0.12)	0.04 (0.08)	0.04 (0.07)
	Scene	0.04 (0.08)	0.16 (0.43)	0.24 (0.56)	0.12 (0.28)	0.04 (0.09)	0.08 (0.18)
	Agents	0.00 (0.03)	0.00 (0.26)	0.04 (0.37)	0.00 (0.02)	0.00 (0.01)	0.00 (0.00)

Table 8. Task success rates and subgoal success rates of various centralized models. Subgoal success rates are given in parentheses. All models are trained with the full training data except GPT-4o.

Tasks	Condition	TeamCraft-VLA-7B	TeamCraft-VLA-13B	GPT-4o	TeamCraft-7B-GridWorld
Building	Test	0.42 (0.81)	0.48 (0.79)	0.00 (0.07)	0.42 (0.88)
	Shape	0.30 (0.75)	0.26 (0.69)	0.00 (0.08)	0.50 (0.90)
	Material	0.30 (0.74)	0.08 (0.63)	0.00 (0.07)	0.26 (0.82)
	Scene	0.40 (0.83)	0.48 (0.83)	0.00 (0.07)	0.48 (0.89)
Clearing	Agents	0.02 (0.57)	0.04 (0.59)	0.00 (0.00)	0.12 (0.71)
	Test	0.64 (0.91)	0.64 (0.94)	0.00 (0.03)	1.00 (1.00)
	Shape	0.56 (0.91)	0.78 (0.96)	0.00 (0.04)	1.00 (1.00)
	Material	0.56 (0.91)	0.56 (0.92)	0.00 (0.12)	1.00 (1.00)
Farming	Scene	0.58 (0.92)	0.48 (0.90)	0.00 (0.06)	1.00 (1.00)
	Agents	0.36 (0.81)	0.16 (0.77)	0.00 (0.00)	0.84 (0.97)
	Test	0.36 (0.64)	0.46 (0.73)	0.00 (0.00)	0.78 (0.86)
	Crop	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
Smelting	Scene	0.38 (0.67)	0.44 (0.75)	0.00 (0.00)	0.90 (0.96)
	Agents	0.38 (0.68)	0.36 (0.72)	0.00 (0.00)	0.40 (0.73)
	Test	0.24 (0.28)	0.32 (0.59)	0.02 (0.02)	0.24 (0.51)
	Goal	0.00 (0.00)	0.00 (0.00)	0.08 (0.08)	0.00 (0.00)
Smelting	Furnace	0.18 (0.20)	0.18 (0.38)	0.00 (0.00)	0.24 (0.39)
	Scene	0.18 (0.23)	0.24 (0.56)	0.00 (0.00)	0.36 (0.58)
	Agents	0.06 (0.13)	0.04 (0.37)	0.00 (0.00)	0.00 (0.31)

Table 9. Comparison of TeamCraft-VLA redundancy rates.

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	0.01	0.02	0.01	0.01	0.01
TeamCraft-VLA-13B-Cen	0.01	0.00	0.01	0.01	0.01
TeamCraft-VLA-7B-Dec	0.13	0.12	0.13	0.24	0.15
TeamCraft-VLA-13B-Dec	0.11	0.11	0.12	0.22	0.14

Table 10. Comparison of TeamCraft-VLA action sequence length.

	Test	Goal	Scene	Agents	Average
TeamCraft-VLA-7B-Cen	6.62	7.63	5.93	6.35	6.63
TeamCraft-VLA-13B-Cen	6.25	7.44	6.46	6.47	6.65
TeamCraft-VLA-7B-Dec	8.42	8.53	8.06	7.38	8.1
TeamCraft-VLA-13B-Dec	8.62	8.46	8.41	6.71	8.04

H. GPT-4o Implementation

We use `gpt-4o-2024-08-06` as the proprietary VLA. Specifically, we use similar prompt structures as the centralized finetuned TeamCraft-VLA model, with additional task information in the initial system prompt as shown in [Figure 17](#), [Figure 19](#) and [Figure 21](#) to provide background

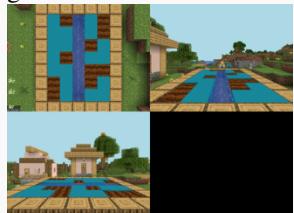
knowledge of the task. The system prompt contains recipes, input, output formats, all available blocks, items, workspace limitations, and one successful rollout of a similar task in the same task family. At the first step, we additionally provide the first user prompt, where the model is given a specific multi-modal task specification accompanied by initial visual observations and inventory details of the agents. Based on the system prompts and user prompts, the model predicts the actions. As the interaction progresses with subsequent prompts, the context is maintained and expanded with the addition of prior responses and updated visual data as shown in [Figure 18](#).

I. GPT-4o Error Analysis

We visualized some failure cases in [Figure 17](#), [Figure 18](#), [Figure 19](#), [Figure 20](#), [Figure 21](#) and [Figure 22](#). In general, GPT4-o fail to understand spatial relations and often chooses the incorrect coordinates as the locations for placing and mining actions. In addition, the model sometimes fails to follow instructions and does not harvest the crop in the farming task, as shown in [Figure 21](#) and [Figure 22](#).

System Prompt

Two bots need to grow on the platform. The goal is to get 10 beetroot. Write the actions for bot1, bot2 based on this given observation."

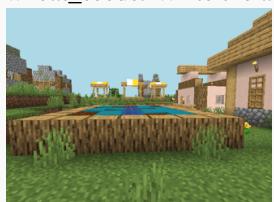


(a) orthographic views of the task

Step 0:

User Prompt

bot1 has 7 beetroot_seeds. bot1 has 1 carrot. bot1 has 3 potato. bot2 has 5 beetroot_seeds. bot2 has 2 wheat_seeds. Write the actions for bot1, bot2 based on this given observation.



(b) Agent1 observation



(c) Agent2 observation

Actions

```
["farm_work(bot1, new Vec3(-1,-1,2), 'sow', 'beet_seeds')",
 "farm_work(bot2, new Vec3(-2,-1,-2), 'sow', 'beet_seeds')"]
```

Step 1:

User Prompt

bot1 has 7 beetroot_seeds. bot1 has 1 carrot. bot1 has 3 potato. bot2 has 5 beetroot_seeds. bot2 has 2 wheat_seeds. Write the actions for bot1, bot2 based on this given observation.



(d) Agent1 observation



(e) Agent2 observation

Actions

```
["farm_work(bot1, new Vec3(3,-1,-1), 'sow', 'beet_seeds')",
 "farm_work(bot2, new Vec3(0,-1,-1), 'sow', 'beet_seeds')"]
```

Figure 12. Object mismatching Example 1.

System Prompt

"Two bots need to craft 2 dried_kelp. here are the instructions: ... To cook a 'dried_kelp', I need a 'kelp'. ... ",



(a) orthographic views of the task

Step 0:**User Prompt**

bot1 has 1 iron_pickaxe. bot1 has 1 iron_axe. bot2 has 2 charcoal. bot2 has 1 iron_shovel. bot2 has 1 iron_sword. Write the actions for bot1, bot2 based on this given observation.



(b) Agent1 observation



(c) Agent2 observation

Actions

```
[putItemFurnace(bot1, 'cobbl1', new Vec3(0,0,-1)),  
putFuelFurnace(bot2, 'charcoal', new Vec3(0,0,-1))]
```

Step 1:**User Prompt**

bot1 has 2 kelp. bot1 has 1 iron_pickaxe. bot1 has 1 iron_axe. bot2 has 1 charcoal. bot2 has 1 iron_shovel. bot2 has 1 iron_sword. Write the actions for bot1, bot2 based on this given observation.



(d) Agent1 observation



(e) Agent2 observation

Actions

```
[putItemFurnace(bot1, 'cobbl1', new Vec3(0,0,-1)),  
putFuelFurnace(bot2, 'charcoal', new Vec3(0,0,-1))]
```

Figure 13. Object mismatching Example 2.

System Prompt

"Four bots need to break everything on the platform." ",



(a) orthographic views of the task

Step 0:

User Prompt

bot0 has 1 stone_axe. bot1 has 1 stone_shovel. bot2 has 1 stone_pickaxe. bot3 has 1 stone_sword. Write the actions for bot0, bot1, bot2, bot3 based on this given observation.



(b) Agent0 observation



(c) Agent1 observation



(d) Agent2 observation



(e) Agent3 observation

Actions

```
[ " mineBlock(bot1, new Vec3(-2,0,1)),  
mineBlock(bot2, new Vec3(-1,0,0)),  
mineBlock(bot3, new Vec3(1,0,1))", ]
```

Step 1:

User Prompt

Write the actions for bot0, bot1, bot2, bot3 based on this given observation."



(f) Agent0 observation



(g) Agent1 observation



(h) Agent2 observation



(i) Agent3 observation

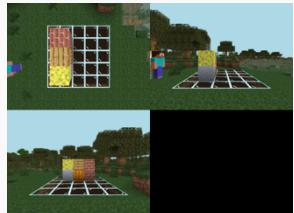
Actions

```
[ " mineBlock(bot1, new Vec3(1,0,0)),  
mineBlock(bot2, new Vec3(-2,0,-2)),  
mineBlock(bot3, new Vec3(1,0,1))", ]
```

Figure 14. Task allocation failure Example 3.

System Prompt

"Four bots need to build a building on the platform. "



(a) orthographic views of the task

Step 0:

User Prompt

bot0 has 4 bricks. bot0 has 5 sponge. bot0 has 2 oak_planks...bot1 has 1 pumpkin... bot2 has 3 bricks... bot3 has 1 clay... Write the actions for bot0, bot1, bot2, bot3 based on this given observation.



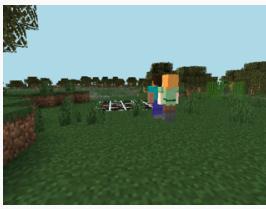
(b) Agent0 observation



(c) Agent1 observation



(d) Agent2 observation



(e) Agent3 observation

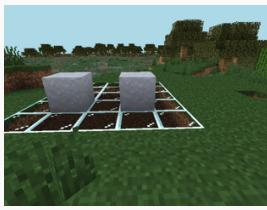
Actions

```
[ " placeItem(bot1, 'bricks', new Vec3(0,0,-1)),  
placeItem(bot2, 'clay', new Vec3(-1,0,-1)),  
placeItem(bot3, 'clay', new Vec3(1,0,-1))" ]
```

Step 1:

User Prompt

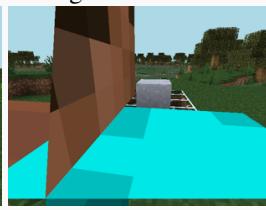
Write the actions for bot0, bot1, bot2, bot3 based on this given observation.



(f) Agent0 observation



(g) Agent1 observation



(h) Agent2 observation



(i) Agent3 observation

Actions

```
[ "placeItem(bot1, 'sponge', new Vec3(-1,1,-1)),  
placeItem(bot2, 'bricks', new Vec3(1,1,-1)),  
placeItem(bot3, 'sponge', new Vec3(0,1,-1))" ]
```

Figure 15. Task allocation failure Example 4.

System Prompt

Two bots need to grow on the platform. The goal is to get 4 carrot.



(a) orthographic views of the task

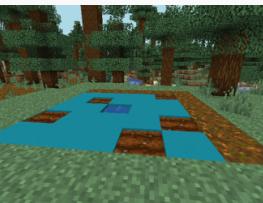
Step 0:

User Prompt

"bot1 has 4 carrot. bot1 has 1 potato. bot1 has 1 beetroot. bot2 has 5 carrot. bot2 has 2 beetroot. bot2 has 3 wheat_seeds. bot2 has 2 potato. Write the actions for bot1, bot2 based on this given observation."



(b) Agent1 observation



(c) Agent2 observation

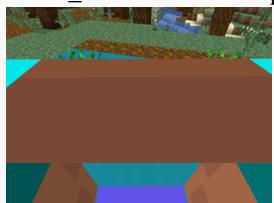
Actions

```
[" farm_work(bot1, new Vec3(1,-1,-2), 'sow', 'carrot'),  
farm_work(bot2, new Vec3(2,-1,-2), 'sow', 'carrot')", ]
```

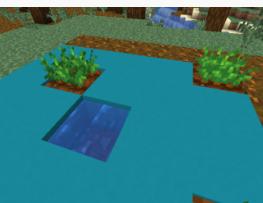
Step 2:

User Prompt

"bot1 has 3 carrot. bot1 has 1 potato. bot1 has 1 beetroot. bot2 has 5 carrot. bot2 has 2 beetroot. bot2 has 3 wheat_seeds. bot2 has 2 potato. Write the actions for bot1, bot2 based on this given observation."



(d) Agent1 observation



(e) Agent2 observation

Actions

```
[" farm_work(bot1, new Vec3(-2,-1,-2), 'sow', 'carrot'),  
farm_work(bot2, new Vec3(-1,-1,1), 'sow', 'carrot')", ]
```

Figure 16. Object state recognition failure Example 5.

System Prompt

You are controlling 3 bots in a Minecraft world. The goal is to build a specific structure on a platform.

Please review the images provided below, which include the current state of the world and the goal structure (the final image is the three orthographic views of the goal). Based on these observations, generate actions for each bot to help build the structure.

Instructions:

- **Action Format:**

- **Bots:**

- 'botID' can be one of: 'bot1', 'bot2', 'bot3', 'bot4' (depending on the number of bots).

- **Blocks:**

- "block" is the type of block to place.

- **Available Blocks:**

- 'oak fence", "birch log", "coal ore", "bricks", "sandstone", "stone", "iron ore", "gold ore", "sponge", "sea lantern", dirt, "grass block", "clay", "oak planks", "emerald block", "pumpkin", "orange concrete", "purple wool", "end stone", "bookshelf", "acacia fence", "oak log".

- **Constraints:**

- **Inventory Awareness:** Ensure each bot has the necessary blocks in their inventory.

- **No Overlapping Blocks:** Do not place more than one block at the same position.

- **Workspace Dimensions:** The center of the workspace is at (0, 0, 0), and it spans 3 units along the x-axis, 3 units along the z-axis, and 2 units along the y-axis.

- **One Action per Bot:** Each bot can place only one block at a time.

Submission Guidelines:

- Provide only the list of action commands for all bots.

- Do not include any additional text, explanations, or formatting (e.g., no code blocks or markdown).

- Example:

```
[ "placeItem(bot1, 'stone', new Vec3(1, 0, 0))", "placeItem(bot2, 'oak planks', new Vec3(0, 0, 1))" ]
```

You need to put "" for each entry in the list.

Please generate the list of commands based on the current observations and the goal image.

User Prompt

Here are the observations at the current time step for each agent. The final image is the goal image. bot1 has 5 dirt. bot1 has 3 clay. bot1 has 7 emerald block. bot1 has 1 oak fence. bot1 has 3 sponge. bot1 has 1 bricks. bot1 has 3 sea lantern. bot2 has 5 bricks. bot2 has 2 sponge. bot2 has 6 sea lantern. bot2 has 2 oak fence. bot2 has 4 emerald block. bot2 has 1 dirt. bot2 has 3 clay. bot3 has 6 emerald block. bot3 has 4 oak fence. bot3 has 2 dirt. bot3 has 2 sponge. bot3 has 3 clay. bot3 has 2 sea lantern.



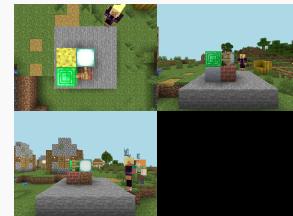
(a) Agent1 observation



(b) Agent2 observation



(c) Agent3 observation



(d) orthographic views of the goal

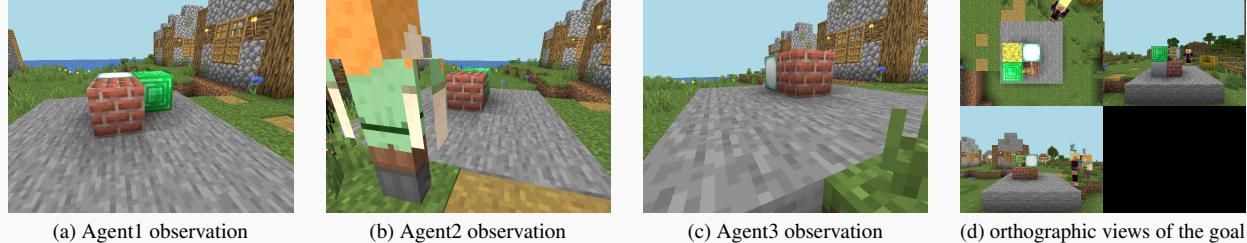
Assistant

```
[placeItem(bot1, emerald block, new Vec3(-1,0,-1)), placeItem(bot2, bricks, new Vec3(0,0,0)),  
placeItem(bot3, sea lantern, new Vec3(-1,0,0))]
```

Figure 17. Sample rollouts of GPT-4o in a building task, Part I.

User Prompt

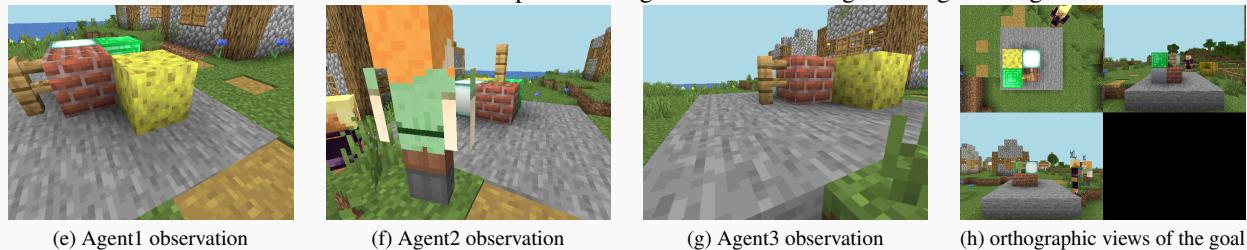
Here are the observations at the current time step for each agent. The final image is the goal image. bot1 has ...

**Assistant**

[placeItem(bot1, sponge, new Vec3(1,0,-1)), placeItem(bot2, oak fence, new Vec3(0,0,1))]

User Prompt

Here are the observations at the current time step for each agent. The final image is the goal image. bot1 has ...

**Assistant**

[placeItem(bot3, sea lantern , new Vec3(0,0,-1))]

Comment

GPT-4o failed to place the blocks in the correct location. The agents placed an additional fence that is not required by the goal. The sponge (yellow) block should be placed next to a sea lattern (white) block.

Figure 18. Sample rollouts of GPT-4o in a building task, Part II.

System Prompt

You are controlling 3 bots in a Minecraft world. The goal is to break all blocks in the world.

Please review the images provided below, which include the current state of the world. Based on these observations, generate actions for each bot to help break the required blocks.

Instructions:

- **Action Format:** - ‘mineBlock(botID, new Vec3(x,y,z))‘ to break a block at the specified coordinates.
- The up axis is the y-axis. x, y, z should be greater than 1.
- **Bots:** - ‘botID‘ can be one of: ‘bot1’, ‘bot2’, ‘bot3’ (depending on the number of bots).
- **Constraints:**
- **Tool Awareness:** Each bot has tools in their inventory to break blocks. Use the ‘item dict durability‘ below to determine how many actions are required based on the tool and block type.
- ‘item dict durability’: item dict durability
- For example, if a bot has a stone axe, it can break a ‘bookshelf’ in 1 action; otherwise, it will take 2 actions.
- **No Duplicate Actions:** Ensure that no bot attempts to break the same block more than once.
- **Workspace Dimensions:** The center of the workspace is at (0, 0, 0), and it spans 5 units along the x-axis (-2, 2), 2 units along the y-axis, and 5 units along the z-axis (-2, 2). So the action can have negative numbers
- **One Action per Bot:** Each bot can attempt to break only one block at a time unless specified by ‘item dict durability’.

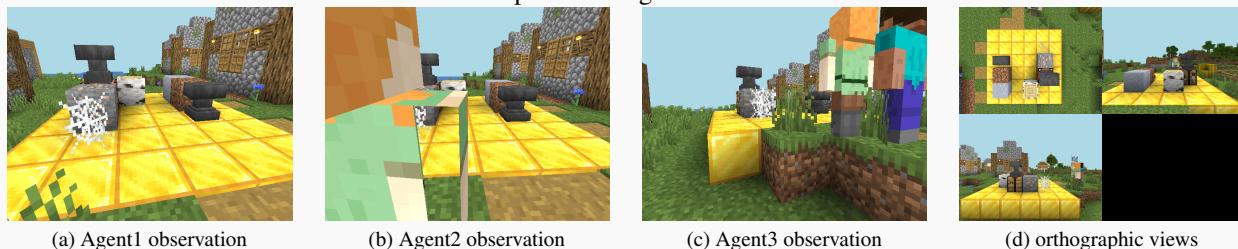
Submission Guidelines:

- Provide only the list of action commands for all bots.
- Do not include any additional text, explanations, or formatting (e.g., no code blocks or markdown).
- Example: [“mineBlock(bot1, new Vec3(1,0,-1))”, “mineBlock(bot2, new Vec3(-1,0,1))”]

Please generate the list of commands based on the current observations, the goal image, and tool durability requirements.
Please generate the list of commands based on the current observations and the goal image.

User Prompt

Here are the observations at the current time step for each agent.



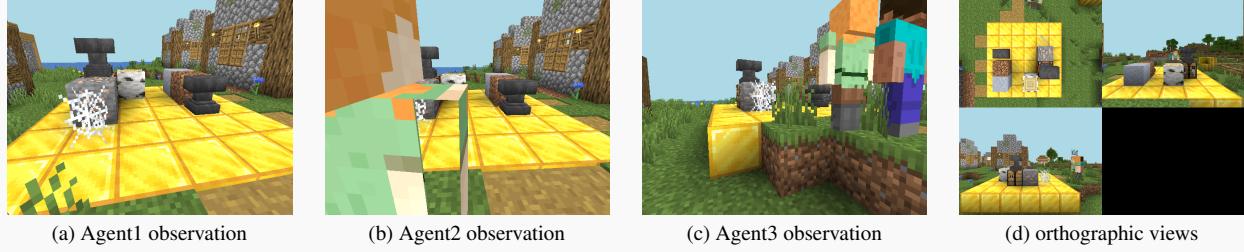
Assistant

[mineBlock(bot1, new Vec3(3,0,2)), mineBlock(bot2, new Vec3(2,0,1))]

Figure 19. Sample rollouts of GPT-4o in a clearing task, Part I.

User Prompt

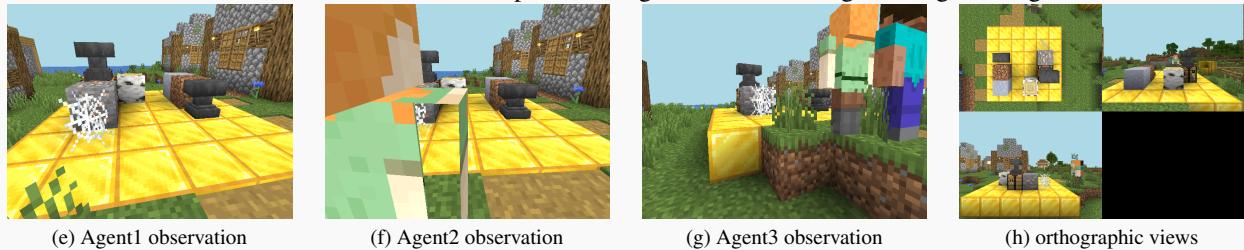
Here are the observations at the current time step for each agent. The final image is the goal image. bot1 has ...

**Assistant**

[mineBlock(bot1, new Vec3(3,0,2)), mineBlock(bot2, new Vec3(2,0,1))]

User Prompt

Here are the observations at the current time step for each agent. The final image is the goal image. bot1 has ...

**Assistant**

[mineBlock(bot1, new Vec3(1,0,2)), mineBlock(bot2, new Vec3(2,0,1))]

Comment

The model did not predict correct block locations and failed to mine any blocks.

Figure 20. Sample rollouts of GPT-4o in a clearing task, Part II.

System Prompt

You are controlling 3 bots in a Minecraft farming environment. The bots are responsible for planting, growing, and harvesting crops like wheat, carrots, and potatoes in a designated farming area.

Please review the images provided below, which include the current state of the farm and the goal farm layout (the final image is the 3-view of the goal). Based on these observations, generate farming commands for each bot to help manage the farming tasks efficiently.

Instructions:

1. **Action Format:** - The actions are in the format: `farmwork(botID, new Vec3(x, y, z), 'action', 'crop')` - 'botID' can be one of: 'bot1', 'bot2', or 'bot3'.

- 'action' is the type of action the bot will perform, such as "sow" (for planting) or "harvest".

- 'crop' is the type of crop the bot will interact with, such as "wheat", "carrot", "potato", or "beetroots".

2. **Bots:** - Each bot has its own inventory and task assignment for sowing seeds, watering, or harvesting the crops.

3. **Crops:** - The available crops are wheat, carrots, and potatoes. Ensure each bot is assigned to the correct action and crop.

4. **Commands:** - Example commands: `["farmwork(bot1, new Vec3(0, 0, -1), 'sow', 'carrot')", "farmwork(bot2, new Vec3(1, 0, -1), 'harvest')"]`

Constraints: - **Inventory Awareness:** Ensure each bot has the necessary items (seeds or tools) in their inventory.

- **No Overlapping Actions:** Do not assign multiple bots to perform the same action on the same crop plot simultaneously.

- **Workspace Dimensions:** The center of the workspace is at (0, 0, 0), and it spans 5 units along the x-axis (-3,3), 5 units along the z-axis(-3,3), and 3 units along the y-axis.

- **One Action per Bot:** Each bot can perform only one action at a time.

Submission Guidelines:

- Provide only the list of farming commands for all bots.

- Do not include any additional text, explanations, or formatting (e.g., no code blocks or markdown).

- Example: `["farmwork(bot1, new Vec3(1, 0, 0), 'sow', 'wheat')", "farmwork(bot2, new Vec3(0, 0, 1), 'harvest')"]`

Please generate the list of commands based on the current observations and the goal farm layout.

*** You need to harvest when the crop is ready to finish the task ***

User Prompt

Here are the observations at the current time step for each agent. Please harvest the crops when they are ready! You can only harvest what you sow. You can only sow seeds on empty plots, and you can only sow what you have in your inventory. When harvesting, do not specify the crop type. Here is the inventory status for each bot: bot1 has 7 xxx



(a) Agent1 observation



(b) Agent2 observation



(c) Agent3 observation



(d) orthographic views

Assistant

`farmwork(bot1, new Vec3(0, 0, -1), 'sow', 'potato')`

`farmwork(bot2, new Vec3(1, 0, 0), 'sow', 'potato')`

`farmwork(bot3, new Vec3(-1, 0, 1), 'sow', 'potato')`

Figure 21. Sample rollouts of GPT-4o in a farming task, Part I.

User Prompt

Here are the observations at the current time step for each agent. bot1 has ...



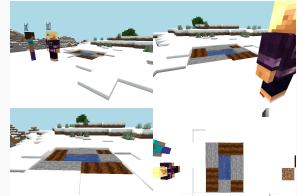
(a) Agent1 observation



(b) Agent2 observation



(c) Agent3 observation



(d) orthographic views

Assistant

```
farmwork(bot1, new Vec3(0, 0, 1), 'sow', 'potato'),  
farmwork(bot2, new Vec3(0, 0, -1), 'sow', 'potato'),  
farmwork(bot3, new Vec3(1, 0, 0), 'sow', 'potato')
```

User Prompt

Here are the observations at the current time step for each agent. bot1 has ...



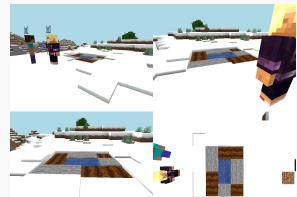
(e) Agent1 observation



(f) Agent2 observation



(g) Agent3 observation



(h) orthographic views

Assistant

```
farmwork(bot1, new Vec3(-1, 0, 0), 'sow', 'potato'),  
farmwork(bot2, new Vec3(1, 0, 0), 'sow', 'potato'),  
farmwork(bot3, new Vec3(0, 0, 1), 'sow', 'potato')
```

Comment

The model does not follow instruction, and keeps predicting the sowing actions without harvesting.

Figure 22. Sample rollouts of GPT-4o in a farming task, Part II.

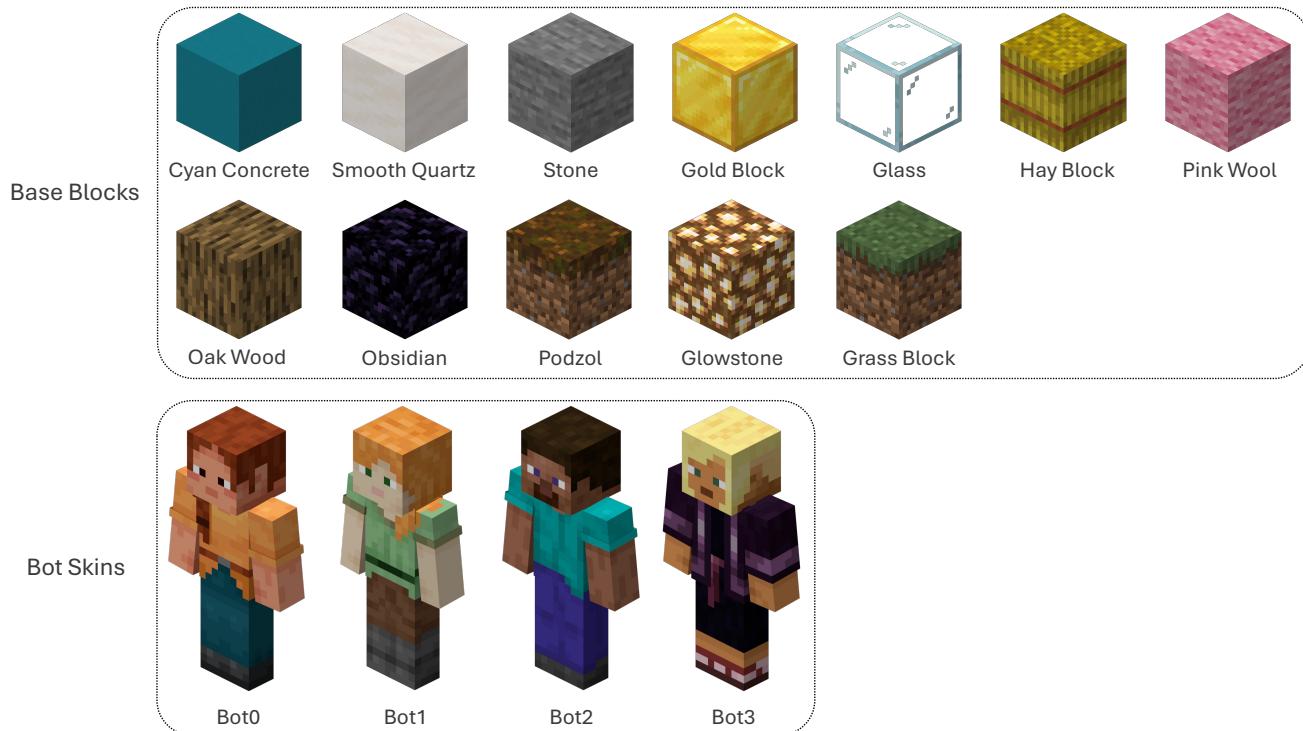


Figure 23. A close-up view of the shared visual diversity in every tasks.

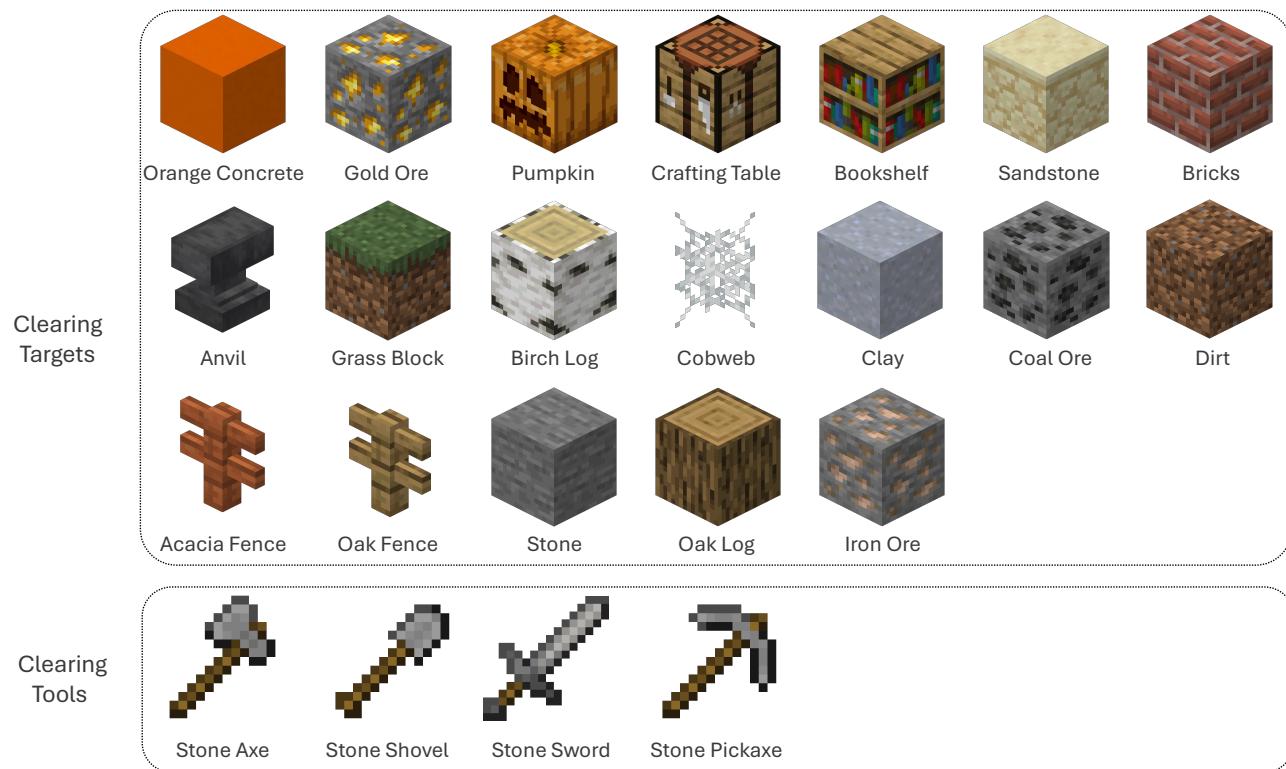


Figure 24. A close-up view of the visual diversity in clearing tasks.



Figure 25. A close-up view of the visual diversity in building tasks.

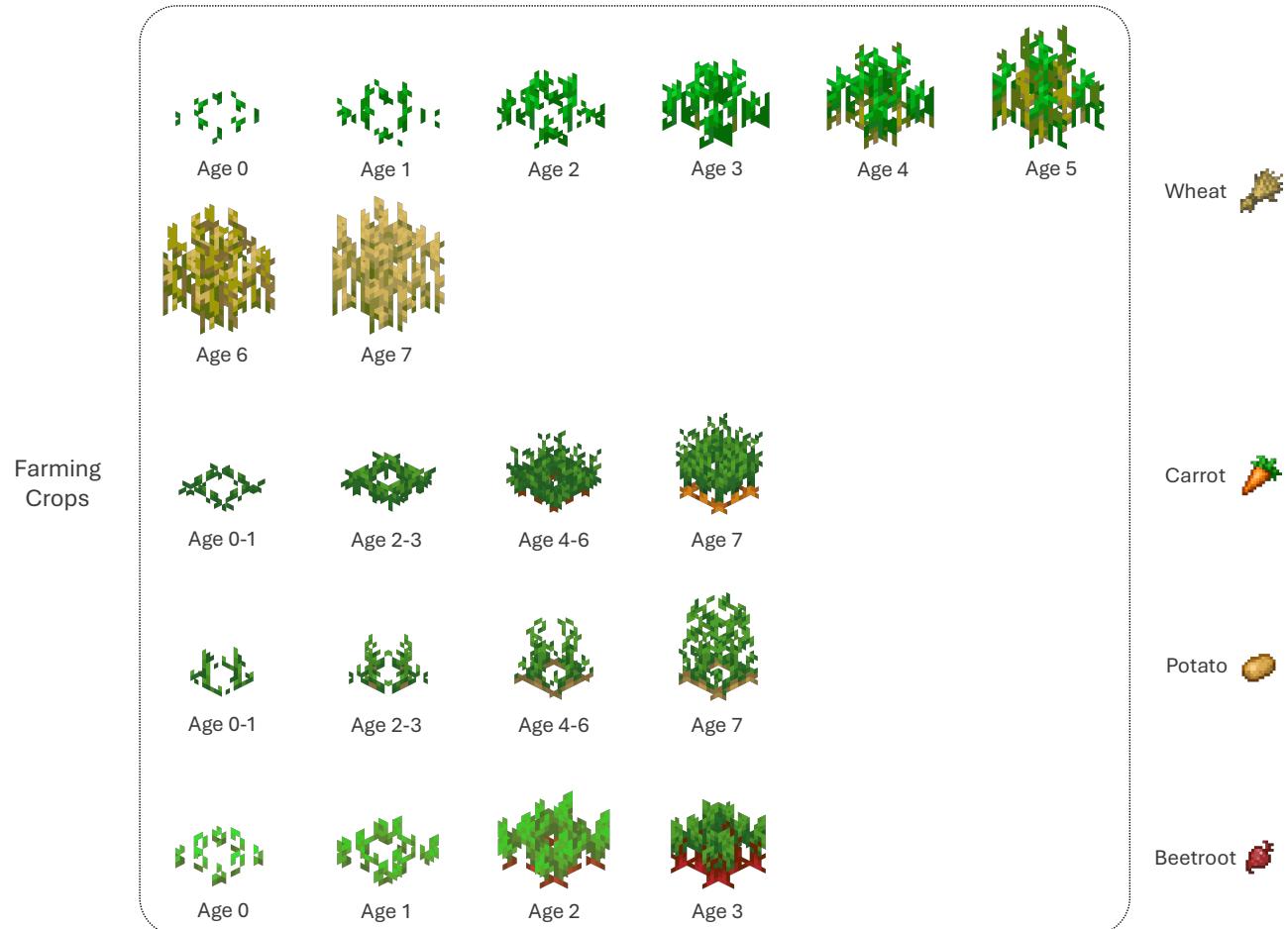


Figure 26. A close-up view of crops appearances across various growing stages in farming tasks.

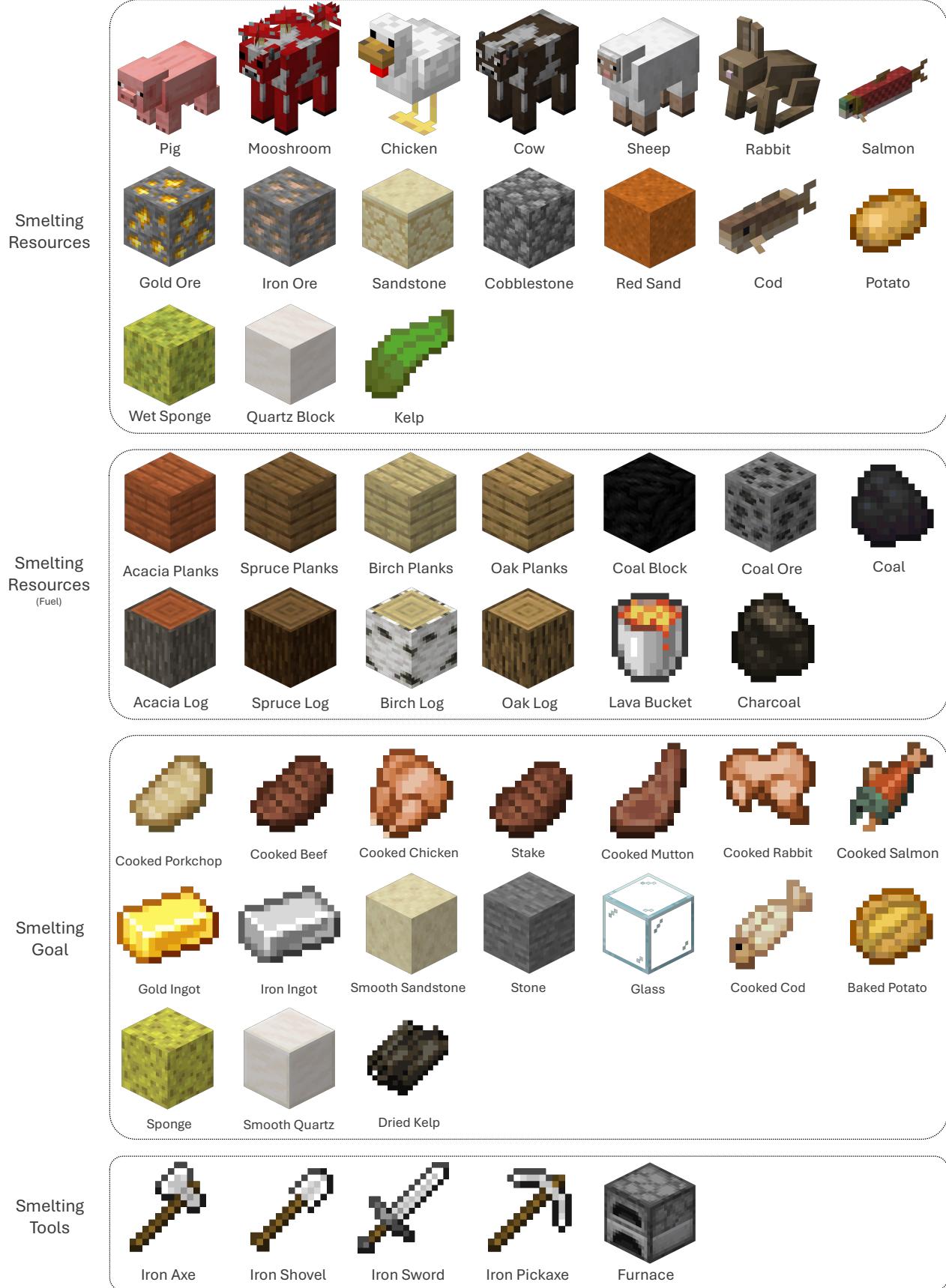


Figure 27. A close-up view of the visual diversity in smelting tasks.



Figure 28. An example scene in the seaside village biome.



Figure 29. An example scene in the grass village biome.

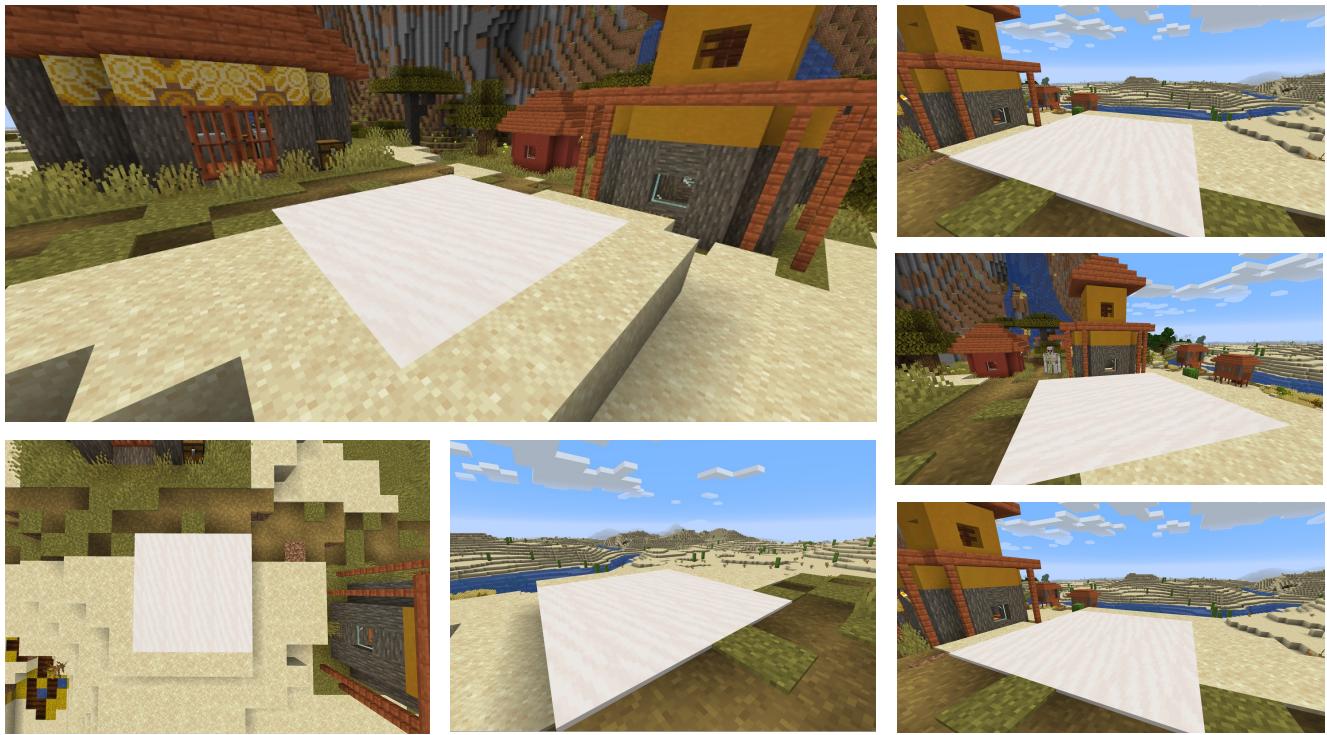


Figure 30. An example scene in the desert village biome.



Figure 31. An example scene in the half mountain biome.

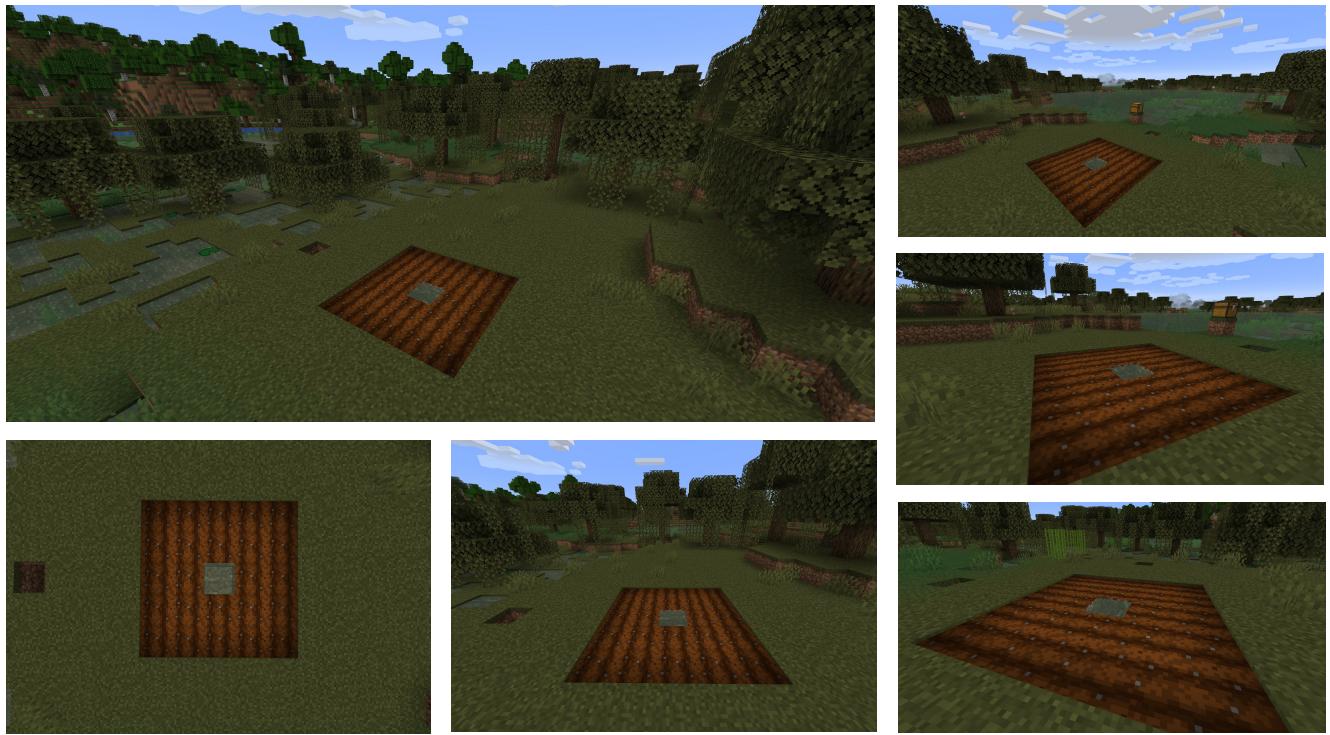


Figure 32. An example scene in the swamp biome.

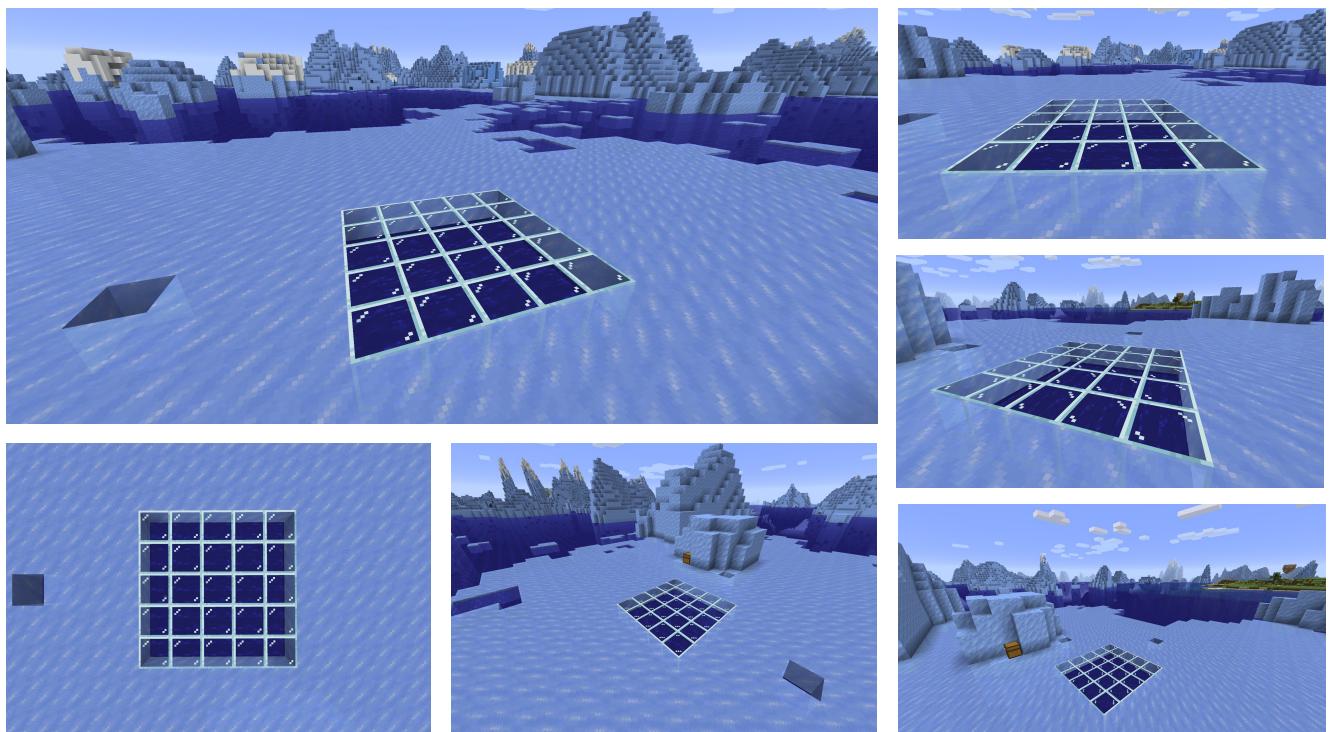


Figure 33. An example scene in the iceberg biome.

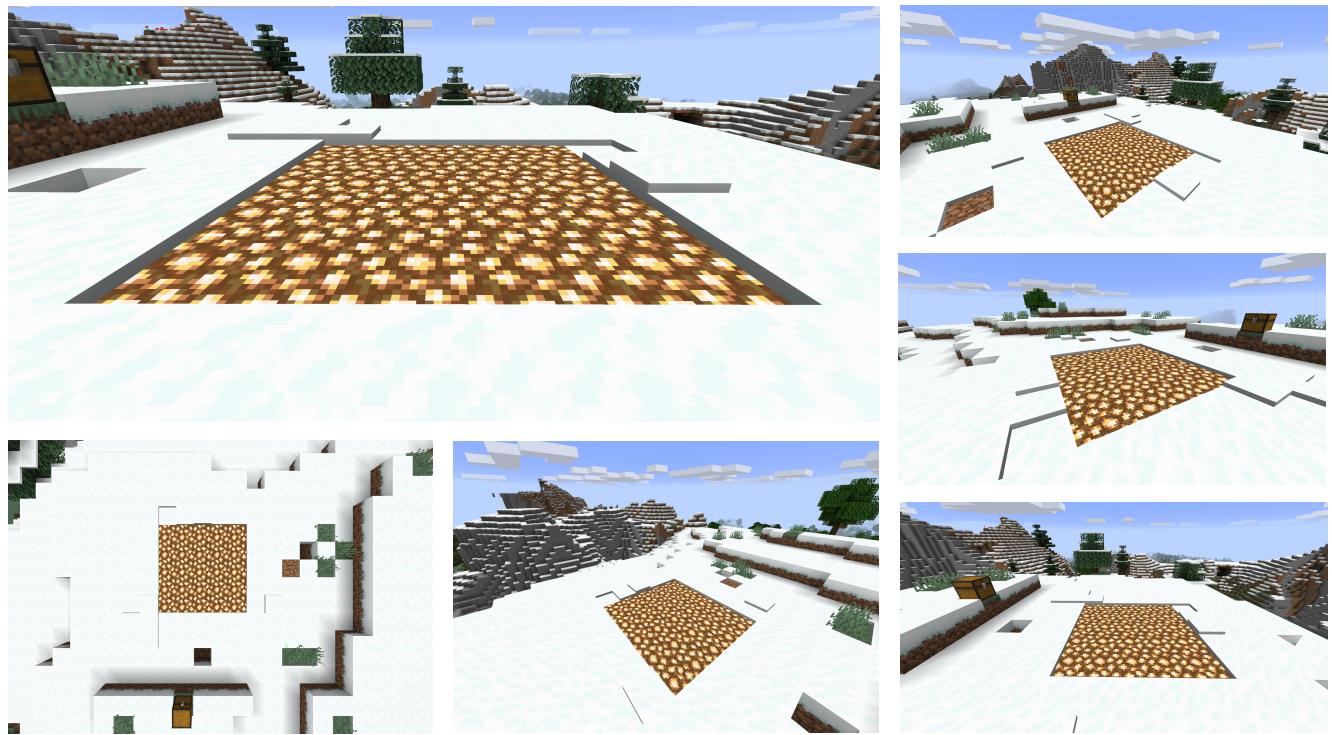


Figure 34. An example scene in the snow mountain biome.

Building 01:

Bot1 Bot2 Bot3 build a building on the playground base halfway up the mountain, following the blueprint given below.

Bot1 has 5 Pumpkins, 2 Sea Lanterns, 1 Coal Ores. Bot2 has 3 Coal Ores, 2 Orange Concrete, 2 Oak Planks. Bot3 has 3 Bricks.

4 Oak Planks. The blueprint looks like:

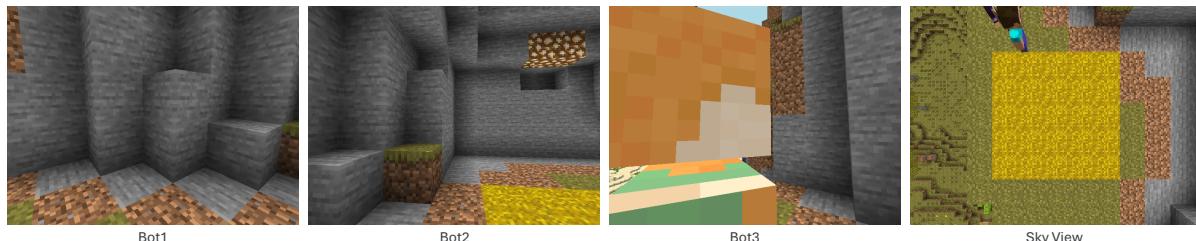
Oak Planks



Step 0:

Actions: None

Reward: 0



Step 1:

Actions: placeItem(bot1, 'pumpkin', new Vec3(-1,0,-1)), placeItem(bot2, 'coal_ore', new Vec3(1,0,-1)), placeItem(bot3, 'bricks', new Vec3(0,0,-1))

Reward: 0.5



Figure 35. An example demonstration in the building task, Part I.

Step 2:

Actions: placeItem(bot1, 'sea_lantern', new Vec3(1,1,-1)), placeItem(bot2, 'orange_concrete', new Vec3(-1,1,-1)), placeItem(bot3, 'oak_planks', new Vec3(0,1,-1))

Reward: 1.0

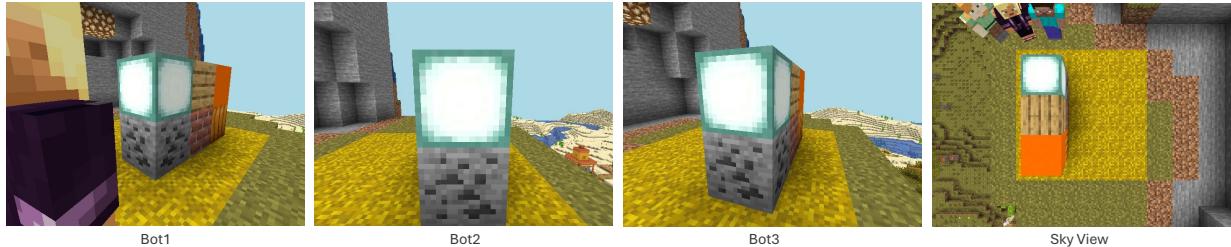
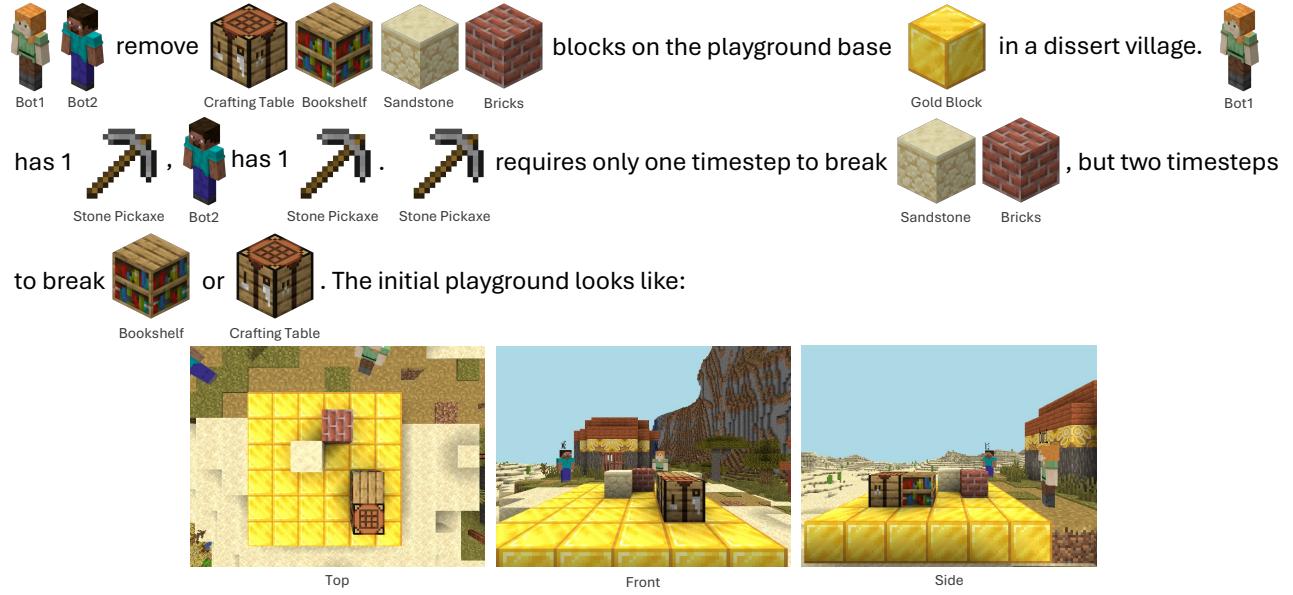


Figure 36. An example demonstration in the building task, Part II.

Clearing 01:



Step 0:

Actions: None

Reward: 0



Figure 37. An example demonstration in the clearing task, Part I.

Step 1:

Actions: mineBlock(bot1, new Vec3(1,0,0)), mineBlock(bot2, new Vec3(0,0,-1))

Reward: 0.5



Step 2:

Actions: mineBlock(bot1, new Vec3(-1,0,1)), mineBlock(bot2, new Vec3(-2,0,1))

Reward: 0.5

Same visual observation as it requires two timesteps to break.

Step 3:

Actions: mineBlock(bot1, new Vec3(-1,0,1)), mineBlock(bot2, new Vec3(-2,0,1))

Reward: 1.0

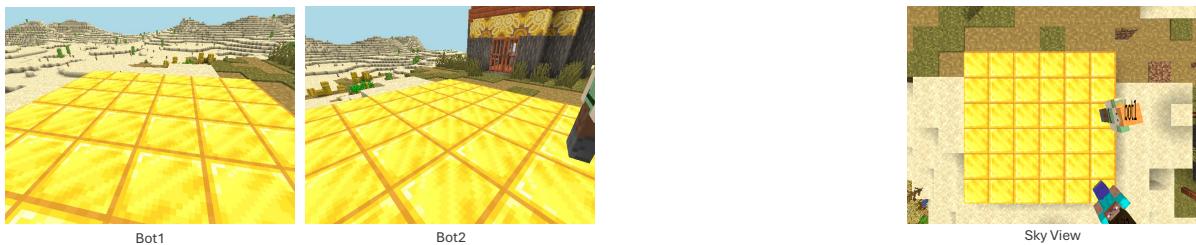


Figure 38. An example demonstration in the clearing task, Part II.

Farming 01:

Bot1 Bot2 sow and harvest for 2 more on on an iceberg. Some are blocked by .

Crops take up to three timesteps to grow from to to be harvestable. Each gives 2 .

Bot1 Carrot Beetroots Bot2 Carrot Potato The initial playground looks like:

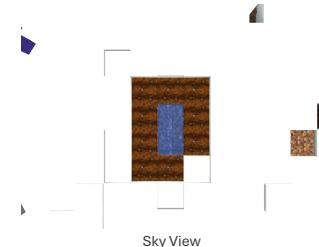
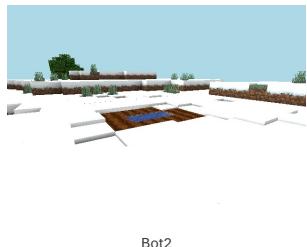
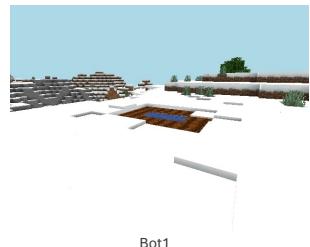


Figure 39. An example demonstration in the farming task, Part I.

Step 0:

Actions: None

Reward: 0



Bot1

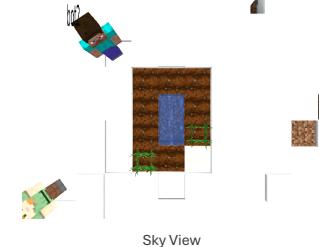
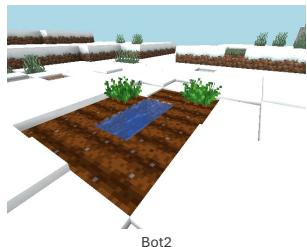
Bot2

Sky View

Step 1:

Actions: farm_work(bot1, new Vec3(-2,-1,-1), 'sow', 'carrot'), farm_work(bot2, new Vec3(-1,-1,1), 'sow', 'carrot')

Reward: 0



Bot1

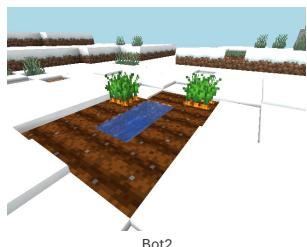
Bot2

Sky View

Step 2:

Actions: None

Reward: 0.5



Bot1

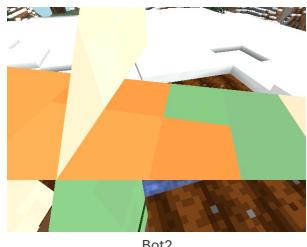
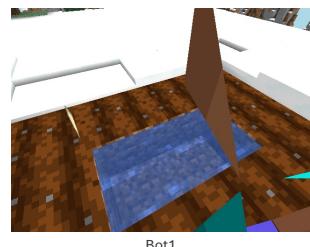
Bot2

Sky View

Step 3:

Actions: farm_work(bot1, new Vec3(-2,0,-1), 'harvest'), farm_work(bot2, new Vec3(-1,0,1), 'harvest')

Reward: 1.0



Bot1

Bot2

Sky View

Figure 40. An example demonstration in the farming task, Part II.

Smelting 01:



Step 0:

Actions: None

Reward: 0



Step 1:

Actions: killMob(bot1, new Vec3(2,0,-3)), putFuelFurnace(bot2, 'birch_log', new Vec3(0,0,0))

Reward: 0



Figure 41. An example demonstration in the smelting task, Part I.

Step 2:

Actions: putItemFurnace(bot1, 'porkchop', new Vec3(0,0,0)), putItemFurnace(bot2, 'porkchop', new Vec3(0,0,0))

Reward: 0.5



Step 3:

Actions: obtainBlock(bot1, new Vec3(-3,0,-3))

Reward: 1.0



Step 4:

Actions: putFuelFurnace(bot1, 'birch_log', new Vec3(0,0,0))

Reward: 1.0



Step 5:

Actions: takeOutFurnace(bot1, new Vec3(0,0,0))

Reward: 1.0

Same visual observation as step 4.

Figure 42. An example demonstration in the smelting task, Part II.

J. Dataset Statistics Tables

Table 11. Building Task Diversity Statistics

Diversity	Type	Count	Percentage
Action Sequences			
3		7,777	51.85%
2		3,207	21.38%
4		3,091	20.61%
5		483	3.22%
6		440	2.93%
Agents			
3		7,505	50.03%
2		7,493	49.97%
Scenes			
	ice_on_water	2,555	17.04%
	mountain_half	2,553	17.03%
	village	2,482	16.55%
	desert_village	2,480	16.53%
	snow_mountain	2,478	16.52%
	swamp	2,450	16.34%
Background Types			
	stone	1,530	10.20%
	pink_wool	1,527	10.19%
	glowstone	1,522	10.15%
	obsidian	1,511	10.08%
	glass	1,509	10.07%
	smooth_quartz	1,499	10.00%
	hay_block	1,494	9.96%
	gold_block	1,473	9.82%
	oak_wood	1,471	9.81%
	cyan_concrete	1,462	9.75%
Target Types			
	bricks	10,391	9.92%
	sponge	5,438	5.19%
	coal_ore	5,370	5.13%
	grass_block	5,327	5.09%
	clay	5,318	5.08%
	sea_lantern	5,296	5.06%
	orange_concrete	5,287	5.05%
	pumpkin	5,269	5.03%
	purple_wool	5,257	5.02%
	gold_ore	5,247	5.01%
	oak_fence	5,234	5.00%
	oak_planks	5,216	4.98%
	birch_log	5,184	4.95%
	stone	5,182	4.95%
	sandstone	5,176	4.94%
	emerald_block	5,164	4.93%
	iron_ore	5,160	4.93%
	dirt	5,124	4.89%
	end_stone	5,119	4.89%
Target Counts			
6		5,653	37.69%
7		2,625	17.50%
8		2,573	17.15%
5		2,122	14.15%
10		526	3.51%
12		515	3.43%
9		496	3.31%
11		488	3.25%
Dimensional Shapes			
	[3, 1, 2]	3,859	25.73%
	[4, 1, 2]	3,770	25.14%
	[2, 3, 2]	3,695	24.63%
	[2, 2, 2]	3,674	24.49%

Table 12. Clearing Task Diversity Statistics

Diversity	Type	Count	Percentage
Action Sequences			
4		4,027	27.51%
5		3,751	25.61%
6		3,270	22.32%
3		1,561	10.66%
7		1,396	9.53%
8		424	2.89%
9		133	0.91%
2		79	0.54%
Agents			
2		7,358	50.28%
3		7,283	49.72%
Scenes			
	desert_village	3,012	20.56%
	snow_mountain	2,948	20.13%
	swamp	2,929	20.00%
	ice_on_water	2,894	19.76%
	village	2,858	19.54%
Background Types			
	smooth_quartz	1,405	9.59%
	pink_wool	1,357	9.27%
	gold_block	1,353	9.24%
	oak_wood	1,334	9.10%
	hay_block	1,332	9.09%
	cyan_concrete	1,332	9.09%
	grass_block	1,328	9.06%
	glass	1,325	9.04%
	glowstone	1,309	8.93%
	stone	1,302	8.89%
	obsidian	1,264	8.63%
Target Counts			
6		4,310	29.43%
5		2,499	17.07%
4		2,436	16.64%
8		1,843	12.58%
7		1,803	12.31%
9		1,750	11.95%
Target Types			
	oak_fence	5,879	6.45%
	grass_block	5,836	6.40%
	clay	5,816	6.38%
	oak_log	5,772	6.33%
	sandstone	5,748	6.30%
	acacia_fence	5,744	6.30%
	birch_log	5,732	6.28%
	bookshelf	5,726	6.28%
	stone	5,709	6.26%
	bricks	5,695	6.25%
	crafting_table	5,684	6.23%
	dirt	5,671	6.22%
	cobweb	5,605	6.15%
	iron_ore	5,603	6.14%
	coal_ore	5,555	6.09%
	anvil	5,439	5.96%
Dimensional Shapes			
3		7,346	50.15%
2		7,295	49.84%
Tools			
	stone_pickaxe	9,329	25.51%
	stone_sword	9,180	25.10%
	stone_axe	9,150	24.99%
	stone_shovel	8,906	24.36%

Table 13. Farming Task Diversity Statistics

Diversity	Type	Count	Percentage
Action Sequences			
4		7,458	50.33%
5		3,731	25.17%
3		3,264	22.02%
6		270	1.82%
2		81	0.55%
7		11	0.07%
Agents			
2		7,465	50.37%
3		7,350	49.63%
Scenes			
snow_mountain		3,732	25.18%
swamp		3,722	25.11%
ice_on_water		3,707	25.01%
village		3,654	24.69%
Background Types			
stone		2,892	19.51%
obsidian		1,549	10.46%
hay_block		1,527	10.30%
oak_wood		1,524	10.28%
cyan_concrete		1,492	10.06%
glass		1,465	9.88%
smooth_quartz		1,462	9.86%
pink_wool		1,455	9.81%
dirt		1,449	9.77%
Target Types			
potato		4,972	33.56%
carrot		4,955	33.45%
wheat		4,888	32.99%
Target Counts			
4		2,873	19.39%
3		2,269	15.31%
5		2,256	15.22%
6		2,151	14.51%
2		1,240	8.37%
8		1,112	7.50%
10		1,062	7.17%
7		933	6.29%
12		512	3.45%
14		407	2.75%

Table 14. Smelting Task Diversity Statistics

Diversity	Type	Count	Percentage
Action Sequences			
5		3,261	30.20%
4		3,072	28.45%
6		2,041	18.89%
3		1,824	16.88%
2		358	3.31%
7		239	2.21%
8		8	0.07%
Agents			
3		5,480	50.75%
2		5,323	49.25%
Scenes			
snow_mountain		2,272	21.04%
desert_village		2,257	20.92%
swamp		2,171	20.08%
ice_on_water		2,059	19.09%
village		2,044	18.87%

Table 14. (cont'd)

Diversity	Type	Count	Percentage
Background Types			
gold_block		1,014	9.22%
smooth_quartz		1,010	9.19%
cyan_concrete		995	9.02%
glowstone		981	8.92%
pink_wool		990	8.99%
glass		978	8.89%
oak_wood		987	8.98%
grass_block		977	8.88%
hay_block		968	8.80%
stone		964	8.76%
obsidian		939	8.54%
Furnace			
1		5,772	53.45%
2		5,031	46.55%
Fuel Types			
coal_block		999	9.58%
charcoal		962	9.22%
lava_bucket		940	9.01%
coal		921	8.84%
spruce_planks		910	8.73%
acacia_planks		906	8.69%
oak_planks		861	8.26%
birch_log		893	8.57%
acacia_log		887	8.50%
spruce_log		845	8.10%
oak_log		840	8.05%
birch_planks		839	8.04%
Goal Types			
food		5,412	50.09%
item		5,391	49.91%
Target Types			
glass		1,144	10.26%
gold_ingot		1,094	9.81%
stone		1,077	9.66%
smooth_sandstone		1,040	9.32%
iron_ingot		1,036	9.29%
cooked_salmon		712	6.38%
cooked_cod		708	6.35%
baked_potato		758	6.80%
cooked_mutton		664	5.95%
cooked_rabbit		648	5.81%
cooked_porkchop		668	5.99%
cooked_beef		627	5.62%
cooked_chicken		627	5.62%
Target Counts			
2		3,999	37.01%
3		3,363	31.13%
1		1,909	17.68%
4		1,532	14.18%
Tools			
iron_pickaxe		18,633	29.69%
iron_shovel		13,676	21.78%
iron_axe		13,453	21.43%
iron_sword		13,448	21.42%
Resource Types			
red_sand		2,032	10.37%
gold_ore		1,999	10.20%
cobblestone		1,915	9.77%
sandstone		1,818	9.28%
iron_ore		1,780	9.08%
coal_ore		1,714	8.75%
acacia_planks		1,564	7.98%
oak_planks		1,503	7.67%
birch_log		1,486	7.58%
spruce_log		1,477	7.54%
oak_log		1,456	7.44%
spruce_planks		1,471	7.51%
birch_planks		1,344	6.86%
sheep		1,119	5.71%
pig		1,104	5.63%
rabbit		1,097	5.60%
chicken		1,081	5.52%
cow		700	3.57%
mushroom		675	3.44%

K. Datasheet

Motivation

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description.

The TeamCraft dataset was created to support development and evaluation for multi-modal multi-agent systems in MineCraft.

Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?

The dataset was created by the TeamCraft team.

Who funded the creation of the dataset? If there is an associated grant, please provide the name of the grantor and the grant name and number.

The dataset was funded by the TeamCraft team.

Any other comments?

None.

Composition

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description.

Each instance contains a ground-truth expert demonstration of a multi-agent team finishing a task in Minecraft, and the corresponding multi-modal prompts specifying the task.

How many instances are there in total (of each type, if appropriate)?

There are in total 57,207 instances.

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

The dataset contain all possible instances.

What data does each instance consist of? “Raw” data (e.g., unprocessed text or images) or features? In either case, please provide a description.

Each instance consists of a multi-modal task specification, agents observations and expert trajectories. Each task specification contains one raw language instruction and three orthographic views images. Agents observations contain the first-person view RGB images and the inventory information.

Is there a label or target associated with each instance? If so, please provide a description.

N/A.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

We intentionally removed the expert demonstration in the test set to prevent over-fitting.

Are relationships between individual instances made explicit (e.g., users’ movie ratings, social network links)? If so, please describe how these relationships are made explicit.

Each instance in the dataset corresponds to an individual task variant that belongs to one of the four task types (i.e. building, clearing, farming, smelting). The task type is explicitly specified in the file name.

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

The dataset has been split into a training set (55,257 instances), a validation set (1,000 instances) and a test set (950 instances). The training set is designed for model training while the validation set is for hyperparameter tuning and checkpoint selection. The test set is designed to evaluate the model’s generalization capabilities across novel scenes, novel goal states and novel agent numbers.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

None as we know.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a dataset

consumer? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The dataset is entirely self-contained.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor– patient confidentiality, data that includes the content of individuals’ non-public communications)? If so, please provide a description.

None as we know.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.

None as we know.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations are identified and provide a description of their respective distributions within the dataset.

No.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

No.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals race or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

None as we know.

Any other comments?

None.

Collection Process

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If the data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe

how.

In each data instance, the expert trajectory was generated programmatically via a planning algorithm. The language instruction was created by language templates. The orthographic views images and agent observations were collected in MineCraft.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatuses or sensors, manual human curation, software programs, software APIs)? How were these mechanisms or procedures validated?

The data is automatically generated by running the data collection scripts. The procedure is further verified by the team via manual inspection.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

N/A.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?

Only the TeamCraft team members are voluntarily involved in the data collection process.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.

The data were collected between February 2024 and September 2024.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

N/A.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

N/A.

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or

otherwise reproduce, the exact language of the notification itself.

N/A.

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.

N/A.

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate).

N/A.

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis) been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

No.

Any other comments?

None.

Preprocessing/cleaning/labeling

Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remaining questions in this section.

Yes. In each data instance, the three orthographic views images rendered by MineCraft are manually concatenated as one image.

Was the “raw” data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the “raw” data.

No.

Is the software that was used to preprocess/clean/label the data available? If so, please provide a link or other access point.

N/A.

Any other comments?

None.

Uses

Has the dataset been used for any tasks already? If so, please provide a description.

The dataset is used to develop the TeamCraft-VLA model, as described in this paper.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.

No.

What (other) tasks could the dataset be used for?

This dataset can be used for the development and evaluation of multi-modal multi-agent systems in MineCraft.

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a dataset consumer might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other risks or harms (e.g., legal risks, financial harms)? If so, please provide a description. Is there anything a dataset consumer could do to mitigate these risks or harms?

Unknown to the authors of the datasheet.

Are there tasks for which the dataset should not be used? If so, please provide a description.

Unknown to the authors of the datasheet.

Any other comments?

None.

Distribution

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.

Yes, the dataset is available on the Internet.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub)? Does the dataset have a digital object identifier (DOI)?

The dataset will be available on [Huggingface](#). It does not have a DOI.

When will the dataset be distributed?

The dataset will be available online by 12/01/2024.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.

The dataset is under Apache 2.0 license.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.

None as we know.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation.

Unknown to authors of the datasheet.

Any other comments?

None.

Maintenance

Who will be supporting/hosting/maintaining the dataset?

The TeamCraft team will be maintaining the dataset.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)?

Email: teamcraftbench@gmail.com

Is there an erratum? If so, please provide a link or other access point.

No.

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to dataset consumers (e.g., mailing list, GitHub)?

No planned updates at the time of preparing this datasheet.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were the individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced.

Unknown to authors of the datasheet.

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to dataset consumers.

N/A.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to dataset consumers? If so, please provide a description.

Others may do so and should contact the original authors about incorporating fixes/extensions.

Any other comments?

None.

References

- [1] A. Amresh, N. Cooke, and A. Fouse, “A Minecraft based simulated task environment for human AI teaming,” in *Proceedings of the 23rd ACM International Conference on Intelligent Virtual Agents*, 2023, pp. 1–3. [9](#)
- [2] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. Van Den Hengel, “Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3674–3683. [2](#)
- [3] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune, “Video Pre-Training (VPT): Learning to act by watching unlabeled online videos,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 639–24 654, 2022. [9](#)
- [4] S. Cai, Z. Wang, K. Lian, Z. Mu, X. Ma, A. Liu, and Y. Liang, “ROCKET-1: Master open-world interaction with visual-temporal context prompting,” *arXiv preprint arXiv:2410.17856*, 2024. [1](#)
- [5] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. Dragan, “On the utility of learning about humans for human-AI coordination,” 2020. [2](#) [9](#)
- [6] M. Chang, G. Chhablani, A. Clegg, M. D. Cote, R. Desai, M. Hlavac, V. Karashchuk, J. Krantz, R. Mottaghi, P. Parashar *et al.*, “PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks,” *arXiv preprint arXiv:2411.00081*, 2024. [2](#)
- [7] B. Chen, S. Song, H. Lipson, and C. Vondrick, “Visual hide and seek,” in *Artificial Life Conference Proceedings 32*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2020, pp. 645–655. [1](#)
- [8] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, “Embodied question answering,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1–10. [2](#)
- [9] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau, “TarMAC: Targeted multi-agent communication,” in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 1538–1546. [1](#)
- [10] Y. Dong, X. Zhu, Z. Pan, L. Zhu, and Y. Yang, “VillagerAgent: A Graph-Based Multi-Agent Framework for Coordinating Complex Task Dependencies in Minecraft,” in *Findings of the Association for Computational Linguistics: ACL 2024*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 16 290–16 314. [Online]. Available: <https://aclanthology.org/2024.findings-acl.964> [1](#) [2](#)
- [11] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar, “MineDojo: Building open-ended embodied agents with internet-scale knowledge,” 2022. [2](#) [9](#)
- [12] Q. Gao, G. Thattai, X. Gao, S. Shakiah, S. Pansare, V. Sharma, G. Sukhatme, H. Shi, B. Yang, D. Zheng *et al.*, “Alexa arena: A user-centric interactive platform for embodied AI,” *arXiv preprint arXiv:2303.01586*, 2023. [2](#)
- [13] X. Gao, Q. Gao, R. Gong, K. Lin, G. Thattai, and G. S. Sukhatme, “DialFRED: Dialogue-enabled agents for embodied instruction following,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 049–10 056, 2022. [2](#)
- [14] X. Gao, R. Gong, Y. Zhao, S. Wang, T. Shu, and S.-C. Zhu, “Joint mind modeling for explanation generation in complex human-robot collaborative tasks,” in *Proceedings of the 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2020, pp. 1119–1126. [9](#)
- [15] R. Gong, X. Gao, Q. Gao, S. Shakiah, G. Thattai, and G. S. Sukhatme, “LEMMA: Learning language-conditioned multi-robot manipulation,” *IEEE Robotics and Automation Letters*, 2023. [2](#)
- [16] R. Gong, J. Huang, Y. Zhao, H. Geng, X. Gao, Q. Wu, W. Ai, Z. Zhou, D. Terzopoulos, S.-C. Zhu, B. Jia, and S. Huang, “ARNOLD: A benchmark for language-grounded task learning with continuous states in realistic 3D scenes,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. [2](#)
- [17] R. Gong, Q. Huang, X. Ma, H. Vo, Z. Durante, Y. Noda, Z. Zheng, S.-C. Zhu, D. Terzopoulos, L. Fei-Fei, and J. Gao, “MindAgent: Emergent gaming interaction,” 2023. [1](#) [2](#)
- [18] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi, “IQA: Visual question answering in interactive environments,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4089–4098. [2](#)
- [19] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov, “MineRL: A large-scale dataset of Minecraft demonstrations,” 2019. [2](#)
- [20] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marrs, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman *et al.*, “Human-level performance in 3D multiplayer games with population-based reinforcement learning,” *Science*, vol. 364, no. 6443, pp. 859–865, 2019. [1](#)
- [21] U. Jain, L. Weihs, E. Kolve, A. Farhadi, S. Lazebnik, A. Kembhavi, and A. Schwing, “A cordial sync: Going beyond marginal policies for multi-agent embodied tasks,” 2020. [1](#) [2](#)
- [22] U. Jain, L. Weihs, E. Kolve, M. Rastegari, S. Lazebnik, A. Farhadi, A. Schwing, and A. Kembhavi, “Two body problem: Collaborative visual task completion,” 2019. [1](#) [2](#) [9](#)
- [23] P. Jayannavar, A. Narayan-Chen, and J. Hockenmaier, “Learning to execute instructions in a Minecraft dialogue,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2589–2602. [2](#) [9](#)
- [24] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, “VIMA: General robot manipulation with multimodal prompts,” *arXiv preprint arXiv:2210.03094*, vol. 2, no. 3, p. 6, 2022. [1](#) [2](#)
- [25] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, “The malmo platform for artificial intelligence experimentation,” in *Ijcai*, 2016, pp. 4246–4247. [2](#)
- [26] J. Z. Leibo, E. Duéñez-Guzmán, A. S. Vezhnevets, J. P. Agapiou, P. Sunehag, R. Koster, J. Matyas, C. Beattie, I. Mordatch, and T. Graepel, “Scalable evaluation of multi-agent reinforcement learning with melting pot,” 2021. [1](#) [2](#)

- [27] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “CAMEL: Communicative agents for “mind” exploration of large language model society,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 51991–52008, 2023. [2](#)
- [28] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual instruction tuning,” *Advances in neural information processing systems*, vol. 36, 2024. [6](#)
- [29] X. Liu, D. Guo, H. Liu, and F. Sun, “Multi-agent embodied visual semantic navigation with scene prior knowledge,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3154–3161, 2022. [2](#)
- [30] X. Liu, X. Li, D. Guo, S. Tan, H. Liu, and F. Sun, “Embodied multi-agent task planning from ambiguous instruction,” *Proceedings of Robotics: Science and Systems, New York City, NY, USA*, pp. 1–14, 2022. [1, 2](#)
- [31] Q. Long, R. Li, M. Zhao, T. Gao, and D. Terzopoulos, “Inverse attention agent for multi-agent system,” 2024. [Online]. Available: <https://arxiv.org/abs/2410.21794> [1](#)
- [32] Q. Long, F. Zhong, M. Wu, Y. Wang, and S.-C. Zhu, “SocialGFs: Learning social gradient fields for multi-agent reinforcement learning,” *arXiv preprint arXiv:2405.01839*, 2024. [2](#)
- [33] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, “Evolutionary population curriculum for scaling multi-agent reinforcement learning,” *arXiv preprint arXiv:2003.10423*, 2020.
- [34] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” 2020. [2](#)
- [35] X. Ma, S. Yong, Z. Zheng, Q. Li, Y. Liang, S.-C. Zhu, and S. Huang, “SQA3D: Situated question answering in 3D scenes,” 2023. [2](#)
- [36] A. Majumdar, A. Ajay, X. Zhang, P. Putta, S. Yenamandra, M. Henaff, S. Silwal, P. Mcvay, O. Maksymets, S. Arnaud et al., “OpenEQA: Embodied question answering in the era of foundation models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16488–16498. [2](#)
- [37] Z. Mandi, S. Jain, and S. Song, “RoCo: Dialectic multi-robot collaboration with large language models,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 286–299. [1, 2, 9](#)
- [38] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, “CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7327–7334, 2022. [2](#)
- [39] I. Mordatch and P. Abbeel, “Emergence of grounded compositional language in multi-agent populations,” *arXiv preprint arXiv:1703.04908*, 2017. [2](#)
- [40] A. Narayan-Chen, P. Jayannavar, and J. Hockenmaier, “Collaborative dialogue in Minecraft,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 5405–5415. [2, 9](#)
- [41] A. Padmakumar, J. Thomason, A. Shrivastava, P. Lange, A. Narayan-Chen, S. Gella, R. Piramuthu, G. Tur, and D. Hakkani-Tur, “TEACH: Task-driven embodied agents that chat,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 2, 2022, pp. 2017–2025. [2](#)
- [42] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023, pp. 1–22. [1, 2](#)
- [43] D. Perez-Liebana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gaina, and D. Ionita, “The multi-agent reinforcement learning in Malmö (MARLÖ) competition,” *arXiv preprint arXiv:1901.08129*, 2019. [1, 2](#)
- [44] X. Puig, T. Shu, S. Li, Z. Wang, Y.-H. Liao, J. B. Tenenbaum, S. Fidler, and A. Torralba, “Watch-And-Help: A challenge for social perception and human-ai collaboration,” 2021. [1, 2](#)
- [45] X. Puig, T. Shu, J. B. Tenenbaum, and A. Torralba, “NOPA: Neurally-guided online probabilistic assistance for building socially intelligent home assistants,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 7628–7634. [2](#)
- [46] M. Shridhar, L. Manuelli, and D. Fox, “CLIPort: What and where pathways for robotic manipulation,” in *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021. [2](#)
- [47] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox, “ALFRED: A benchmark for interpreting grounded instructions for everyday tasks,” 2020. [2](#)
- [48] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht, “ALFWorld: Aligning text and embodied environments for interactive learning,” *arXiv preprint arXiv:2010.03768*, 2020. [2](#)
- [49] L. Smith and M. Gasser, “The development of embodied cognition: Six lessons from babies,” *Artificial life*, vol. 11, no. 1-2, pp. 13–29, 2005. [1](#)
- [50] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” *Autonomous Robots*, vol. 8, pp. 345–383, 2000. [1](#)
- [51] J. Suárez, D. Bloomin, K. W. Choe, H. X. Li, R. Sullivan, N. Kanna, D. Scott, R. Shuman, H. Bradley, L. Castricato et al., “Neural MMO 2.0: A massively multi-task addition to massively multi-agent learning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024. [2](#)
- [52] J. Suarez, Y. Du, C. Zhu, I. Mordatch, and P. Isola, “The neural MMO platform for massively multiagent research,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung, Eds., vol. 1, 2021. [1, 2](#)
- [53] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets et al., “Habitat 2.0: Training home assistants to rearrange their habitat,” *Advances in neural information processing systems*, vol. 34, pp. 251–266, 2021. [2](#)
- [54] S. Tan, W. Xiang, H. Liu, D. Guo, and F. Sun, “Multi-agent embodied question answering in interactive environments,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*. Springer, 2020, pp. 663–678. [2](#)

- [55] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019. [2](#)
- [56] Y. Wan, J. Mao, and J. Tenenbaum, “HandMeThat: Human-robot communication in physical and social environments,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 014–12 026, 2022. [2](#)
- [57] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, “Voyager: An open-ended embodied agent with large language models,” 2023. [1](#), [2](#)
- [58] Z. Wang, S. Cai, A. Liu, Y. Jin, J. Hou, B. Zhang, H. Lin, Z. He, Z. Zheng, Y. Yang, X. Ma, and Y. Liang, “Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models,” *arXiv preprint arXiv:2311.05997*, 2023. [9](#)
- [59] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, “Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents,” 2023. [9](#)
- [60] S. A. Wu, R. E. Wang, J. A. Evans, J. B. Tenenbaum, D. C. Parkes, and M. Kleiman-Weiner, “Too many cooks: Bayesian inference for coordinating multi-agent collaboration,” *Topics in Cognitive Science*, vol. 13, no. 2, pp. 414–432, 2021. [1](#)
- [61] C. Yu, A. Velu, E. Vinitksy, J. Gao, Y. Wang, A. Bayen, and Y. Wu, “The surprising effectiveness of PPO in cooperative multi-agent games,” 2021. [2](#)
- [62] X. Yu, J. Fu, R. Deng, and W. Han, “MineLand: Simulating large-scale multi-agent interactions with limited multimodal senses and physical needs,” 2024. [1](#)
- [63] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attaran, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee, “Transporter networks: Rearranging the visual world for robotic manipulation,” *Conference on Robot Learning (CoRL)*, 2020. [2](#)
- [64] C. Zhang, K. Yang, S. Hu, Z. Wang, G. Li, Y. Sun, C. Zhang, Z. Zhang, A. Liu, S.-C. Zhu, X. Chang, J. Zhang, F. Yin, Y. Liang, and Y. Yang, “ProAgent: Building proactive cooperative agents with large language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 591–17 599. [1](#)
- [65] H. Zhang, W. Du, J. Shan, Q. Zhou, Y. Du, J. B. Tenenbaum, T. Shu, and C. Gan, “Building cooperative embodied agents modularly with large language models,” in *Proceedings of the Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=EnXJfQqy0K> [2](#)
- [66] H. Zhang, Z. Wang, Q. Lyu, Z. Zhang, S. Chen, T. Shu, Y. Du, and C. Gan, “COMBO: Compositional world models for embodied multi-agent cooperation,” *arXiv preprint arXiv:2404.10775*, 2024. [1](#)
- [67] K. Zheng, X. Chen, O. C. Jenkins, and X. Wang, “VLMbench: A compositional benchmark for vision-and-language manipulation,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 665–678, 2022. [2](#)