Your Co-Workers Matter: Evaluating Collaborative Capabilities of Language Models in Blocks World

Anonymous ACL submission

Abstract

Language agents that interact with the world on their own have great potential for automating digital tasks. While large language model (LLM) agents have made progress in understanding and executing tasks such as textual games and webpage control, many real-world tasks also require collaboration with humans or other LLMs in equal roles, which involves intent understanding, task coordination, and communication. To test LLM's ability to collaborate, we design a blocks-world environment, where two agents, each having unique goals and skills, build a target structure together. To complete the goals, they can act in the world and communicate in natural language. Under this environment, we design increasingly challenging settings to evaluate different collaboration perspectives, from independent to more complex, dependent tasks. We further adopt chain-of-thought prompts that include intermediate reasoning steps to model the partner's state and identify and correct execution errors. Both human-machine and machine-machine experiments show that LLM agents have strong grounding capacities, and our approach significantly improves the evaluation metric.

1 Introduction

As large language models (LLMs) evolve, they are increasingly expected to collaborate closely with humans or other LLM agents, emphasizing the importance of coordination and communication (Radford et al., 2019). For instance, an LLM assistant might need to work alongside human professionals to plan tasks, manage projects, and ensure efficient execution. While multi-agent collaboration is not a new area (Leibo et al., 2021; Carroll et al., 2019; He et al., 2017), there is limited study on how LLM agents collaborate with humans in *equal* roles, rather than passively following human's instructions.

To fill this gap, we introduce a collaborative

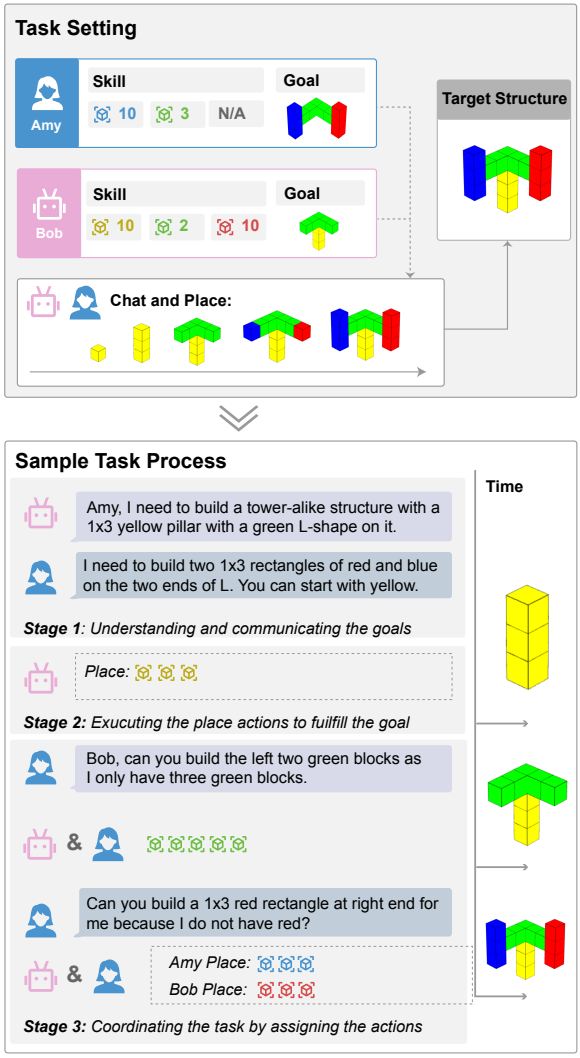

Figure 1: **Task Setting:** A human agent (Amy) and an LLM agent (Bob) collaborate on building the block structure with diverse goals and inventories. **Sample Task Process:** In the shown task, Amy's goal relies on Bob's, so they have to coordinate. To succeed on this task, Amy and Bob have to 1) communicate their goals and figure out the immediate plan to complete; 2) Amy *place* the yellow blocks to complete the immediate plan; 3) Amy and Bob coordinate to complete the remaining part of their goals.

blocks world environment (COBLOCK), to evaluate the collaboration ability of LLMs. In this environment, two agents (either human or LLM), each equipped with complementary goals and skills, collaborate to build a target structure without a designated leader. For example, as shown in Figure 1, Amy is responsible for the top structure, which must be built on top of the base structure that Bob needs to build first. Meanwhile, Amy does not have the red blocks that are necessary to complete her goal. Therefore, they must coordinate their actions, like deciding what to build first and who builds what (e.g., Amy may ask Bob to build the yellow pillar first).

We design three scenarios to progressively increase the level of interaction and coordination required between the two agents, moving from independent tasks to more complex, interdependent tasks (Figure 2).

- **Independent tasks**: Each agent can complete its assigned partial structure without the partner's participation.
- **Skill-dependent tasks**: One agent relies on the other's "skill" or resources to complete the task. For instance, Bob needs Amy's help to build the yellow bar because he doesn't have any yellow blocks.
- **Goal-dependent tasks**: This scenario involves dependence between the agents' assigned partial structure (i.e. goal), requiring more advanced planning and coordination.

To ground LLMs in COBLOCK and enable them to complete the tasks, we prompt GPT series models (GPT-3.5 and GPT-4) to make decisions about the next action given the current observation. At each round, our basic prompt describes the agent's goal (the color and position of every block), the current state (the structure built so far, the action and message history), and candidate actions to choose from (place a block, send a message, wait or terminate the task).

To guide the model in choosing the right action, we additionally include chain-of-thought prompts (CoT) (Wei et al., 2022) to go through intermediate reasoning steps: (1) analysis of the currently built structure and the agent's goal; (2) partner-state modeling to predict the current intent and state of the partner; and (3) self-reflection that identifies and corrects any execution error, and adjusts communication style accordingly.

We evaluate the LLM agents in COBLOCK under both machine-machine and human-machine settings, measured by task success rate, total steps, and workload balance. First, we find that LLM agents have a high success rate when building the structure independently, demonstrating strong grounding ability even without multi-modal training. Second, adding partner-state modeling and self-reflection prompting results in a 30% absolute increase in task completion rate and leads to better workload balance. Finally, while human-machine collaboration has a slightly higher success rate than machine-machine collaboration, humans often take on more responsibility in challenging scenarios when the LLM agent struggles. We hope our findings and the evaluation environment will support future studies on communication and coordination in multi-agent collaboration.[1]

2 The Collaborative Blocks World

In this section, we describe the collaborative blocks world environment COBLOCK (Section 2.1) and the three types of collaboration tasks (Section 2.2).

2.1 The COBLOCK Environment

As shown in Figure 1, in each task, two agents work together to build a target structure by taking actions in the COBLOCK. Specifically, either agent takes one action in each round, we move to the next round after actions from both agents are executed. The task is terminated when the target structure is completed or either agent takes the *end_task* action.

Structure. A *structure* G consists of a set of blocks $B = \{b_1, b_2, \cdots, b_m\}$, where each block b is specified by a color c from a set of colors C (we use a total of six colors), and a 3D position given by coordinates (x, y, z). To simulate real-world conditions, a structure must adhere to the gravity restriction, where each block must be adjacent to at least one other block or be on the ground plane (i.e. $y = 0$). A *sub-structure* of G is a smaller structure formed from a subset of G's blocks.

Agent. Each agent is given a *goal* g, which is a sub-structure of the target structure G. The union of the sub-structures forms the target structure. For example, in Figure 1, Amy is given the upper part of the target structure while Bob is given the lower pillar. Additionally, each agent has an *inventory* $e = \{(c_1, n_1), \cdots, (c_k, n_k)\}$, where n_i denotes

[1] We will release all code and data upon acceptance.

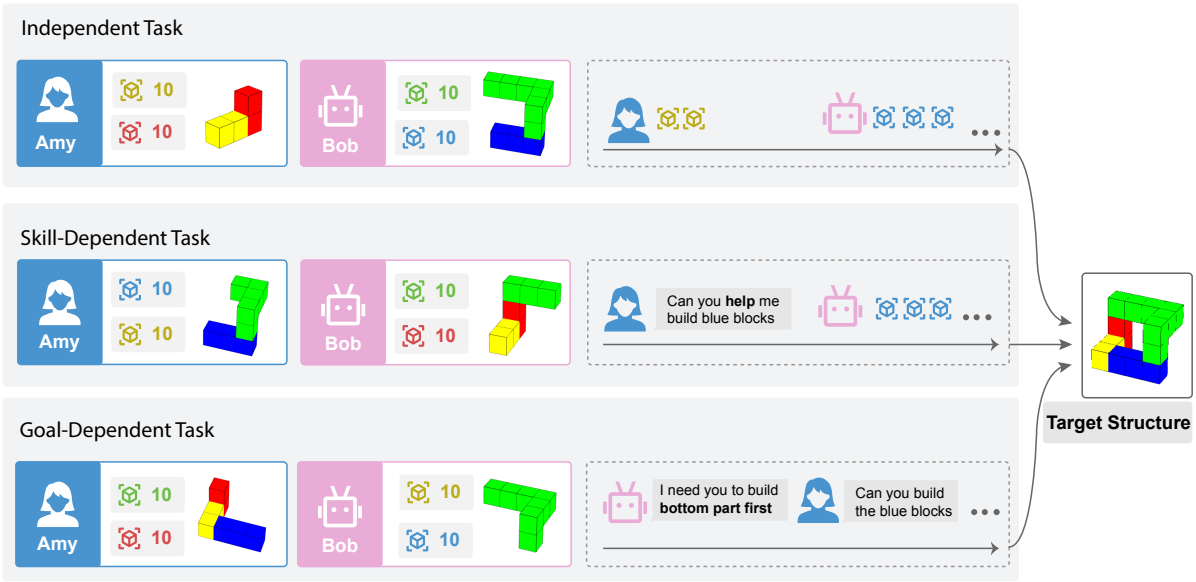

Figure 2: Three different collaboration tasks with increasing levels of coordination. Top: The independent tasks that require little coordination between agents; Middle: The skill-dependent tasks that at least one goal requires both agents to complete; Bottom: The goal-dependent tasks that one agent's goal depends on prior completion of the partner's goal.

the number of blocks of color c_i. Each inventory contains a subset of colors present in G. The goal and inventory of each agent are private so they must communicate to figure out who needs what.

State. The state of the world consists of all public information, including the structure built so far, and the history of the agents' utterances and actions.

Actions. An agent can take one of the following actions: *place* a block from its inventory at specific coordinates; *remove* a block that was placed at specific coordinates (the block will not be used anymore); *send messages* to its partner; *wait* without performing any action; and *terminate* the task.

2.2 Collaboration Tasks

To evaluate different aspects of collaboration, we design three types of tasks with increasing levels of dependence between two agents. We ensure there exists at least one solution to complete the task. Examples of the tasks are shown in Figure 2.

Independent tasks. These tasks require minimal coordination, as two agents can build their goals separately to complete the target structure. However, the agents still need to communicate initially to determine that they can each build their substructures independently.

Skill-dependent tasks. An agent cannot complete its goal without assistance from its partner, as it lacks certain colors of blocks. For example, in Figure 2, Amy needs to describe the green substructure to Bob so that Bob can build it using his green blocks. To succeed in these tasks, an agent must communicate its needs, understand the partner's needs, and then coordinate the actions accordingly.

Goal-dependent tasks. In these tasks, in addition to the dependence between agents' skills, an agent's ability to complete their goal may depend on prior completion of the partner's goal due to the gravity restriction. For example, in Figure 2, Amy must build her sub-structure first (which requires Bob's help with the yellow blocks), so that Bob can build his sub-structure on top of it later. These tasks are the most challenging, as there are complex dependencies among the steps to finish the target structure.

3 Building Collaborative Agents based on LLMs

We use an LLM as the base agent and query it at each round to predict the next action given the current world state (e.g., the structure built so far and the agent's goal).

- **Input:** The input consists of descriptions of the state of the world described in XML (Figure 3 left), including the goal of each agent, the structure built so far, the dialogue and action histories. We use a sequence of blocks (with color and 3D position) to represent the structures.
- **Output:** The output is the next action (*place, break, send message, wait or terminate*). We instruct the model to use a specific format (e.g., `send_message(message="Hello")`) to ensure that it can be parsed for execution in the environment.

To help the LLM agent decide the next action, we use CoT prompting to guide the LLM agent through several intermediate steps designed to anticipate the partner's needs and adjust the construction plan given new observations. We describe each step below and show examples in Figure 3.

Step 1: understanding the world state. The world state is represented using the XML format. Therefore, we prompt LLM agent to first parse the state and describe it in natural language. For example, given the `<goal>` and `<built structure>` field, the agent infers that it should build the yellow blocks next.

Step 2: modeling the partner's state. Effective collaboration requires an agent to understand the partner's needs and coordinate with them to strike a balance between assisting others and fulfilling their own goal. For example, in skill- and goal-dependent tasks, the agent must identify which sub-structures their partner needs help with and integrate it into the planning of their goals. Therefore, we incorporate *partner-state modeling* (Boutilier, 1996; Chalkiadakis and Boutilier, 2003) into our prompts as an intermediate step. As shown in Figure 3, based on the dialogue and action history, the agent predicts the intent and state of its partner, including the goal, the inventory set, the immediate next plan (e.g., building the green sub-structure), and whether the plan has been executed.

Step 3: reflecting on past actions and dialogue. Misunderstandings in collaboration can lead to incorrect actions, and if these errors aren't corrected promptly, they may result in higher expenses to rectify them later. For instance, an agent might place a block of the wrong color at a specified position. If this error isn't corrected in the subsequent round, it could take multiple backtracking actions to correct the mistake. To address this issue, we adopt the *self-reflection mechanism* (Fu et al., 2023) to help the agent review the currently built structure and correct low-level construction errors and high-level communication strategies.

Specifically, we compare the built structure and the agent's goal to check if there is an error using an external program (i.e. whether the built structure is exactly a partial structure of the goal). The feedback is then sent to the agent to correct the potential error. In addition, we instruct the agent to adjust their communication strategies (Hovland et al., 1953) based on the feedback along three dimensions: team role (whether to take the lead and assign tasks to the partner), altruism/egoism balance (whether to pursue their own goals or assist the partner), and persuasion strategy (whether to be more proactive and persuasive in communication).

Step 4: predicting the next action. The final step generates the next action based on the world state, partner state, and the reflection result in previous steps.

Our final prompt starts with a general description of the game and the input/output format for the model to understand the XML and the action functions. To guide the LLM agent to decide the next action through the above reasoning steps, we then provide a CoT prompt that consists of a sequence of reasoning examples, where each example consists of the current world state and the four reasoning steps. A full prompt is shown in Appendix B.

4 Experiments

Successfully completing tasks in CoBlock requires both grounding and collaboration abilities. Therefore, we first access whether LLMs are able to execute instructions in CoBlock correctly through a **single-agent experiment** (Section 4.2). Next, we evaluate whether LLM agents can effectively collaborate with other LLM or human agents through communication and coordination using a **multi-agent experiment** (Section 4.3 and 4.4).

4.1 Implementation Details

Environment. We implement the CoBlock environment using a web interface based on React.js (Facebook Inc., 2023), and render the 3D world through THREE.js (Danchilla and Danchilla, 2012). To generate target structures, we follow five

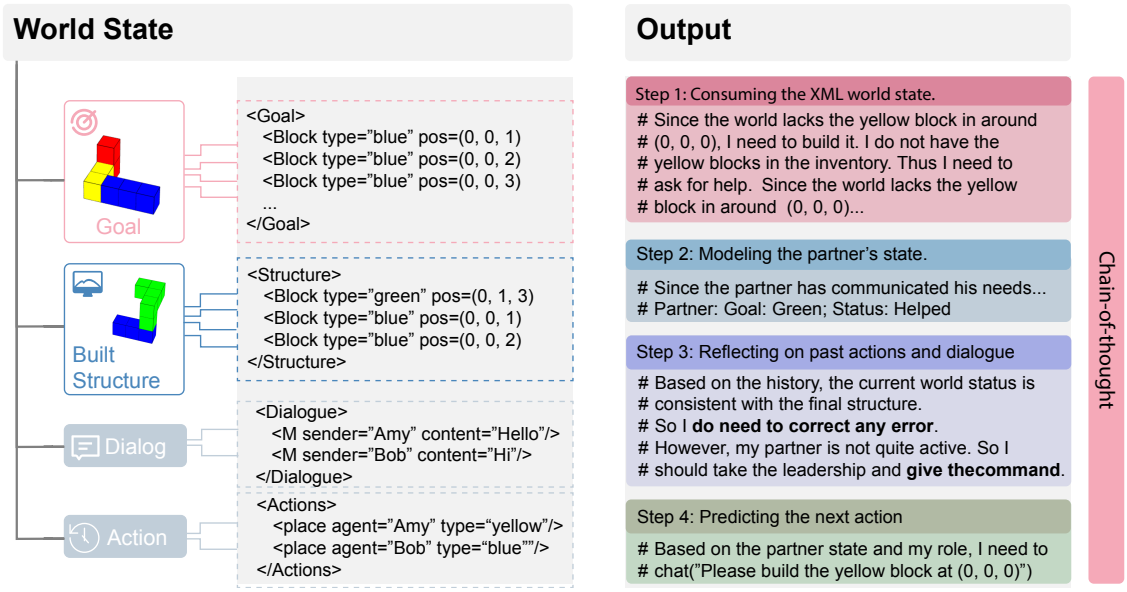

Figure 3: **World State** consists of the agent's goal, currently built structure, dialogue, and action history. **Prompt Text** consists of four steps: 1) Analyze the XML world state and summarize the useful information; 2) Infer both the agent and the partner's state; 3) Self-reflection which identifies the errors and adjusts the communication strategies; 4) Predict the action. We use the CoT prompts in all steps.

common structures used in (Narayan-Chen et al., 2019): symbol, bridge, arch, tower, and rectangle. We then use depth-first search to find combination of these common structures and manually adjusted them if needed. To generate the goals and skills for each agent, we manually split the target structures into two sub-structures and assign inventories to make sure that the tasks can be completed collaboratively. In total, we create 24 collaboration tasks in CoBlock. The details of task creation can be found in Appendix A.1.

Agents. We recruit human participants with experience playing Minecraft (which is similar to CoBlock) via Amazon Mechanical Turk. Participants must complete a tutorial and pass a qualification exam to perform the tasks. We pay participants with an hourly rate of $18. Based on our pilot study, the tasks are usually completed within thirty minutes. For LLM agents, we provide eight in-context CoT examples covering three types of tasks: two independent tasks, three skill-dependent tasks, and three goal-dependent tasks. we also include a baseline LLM agent, using CoT prompts but without partner-state modeling (Step 2) and self-reflection mechanism (Step 3).

4.2 Single-Agent Experiment

We propose three single-agent tasks to evaluate the grounding abilities of LLM agents (Figure 4): 1) Generate textual descriptions from XML-represented structures; We manually evaluate these descriptions to verify the LLM agent's ability to describe the target structure. 2) Generate a sequence of actions given the XML structures; This task measures the agent's ability for task planning. 3) Generate the action sequence from the textual descriptions. This task combines the first two tasks and therefore requires both grounding and planning capacities.

Evaluation Metrics. We use **success rate** to evaluate the proportion of tasks completed by agents successfully (i.e. generate the correct description in task 1 and the action plan in tasks 2 and 3):

$$\text{Success Rate} = \frac{\text{Number of Success}}{\text{Total Number of Tasks}} \quad (1)$$

Results. According to Figure 5, both GPT-4 and GPT-3.5 agents successfully complete almost all three tasks, which indicates that LLM agents have the sufficient grounding and planning skills for the collaborative blocks world. Therefore, the observed failures in the collaboration tasks (which we present next) should not mainly come from the grounding and planning capacities of agents.

4.3 Multi-Agent Experiment Setup

We propose two multi-agent task settings: **Human-machine collaboration** that a human agent and an LLM agent build the structures. This setting

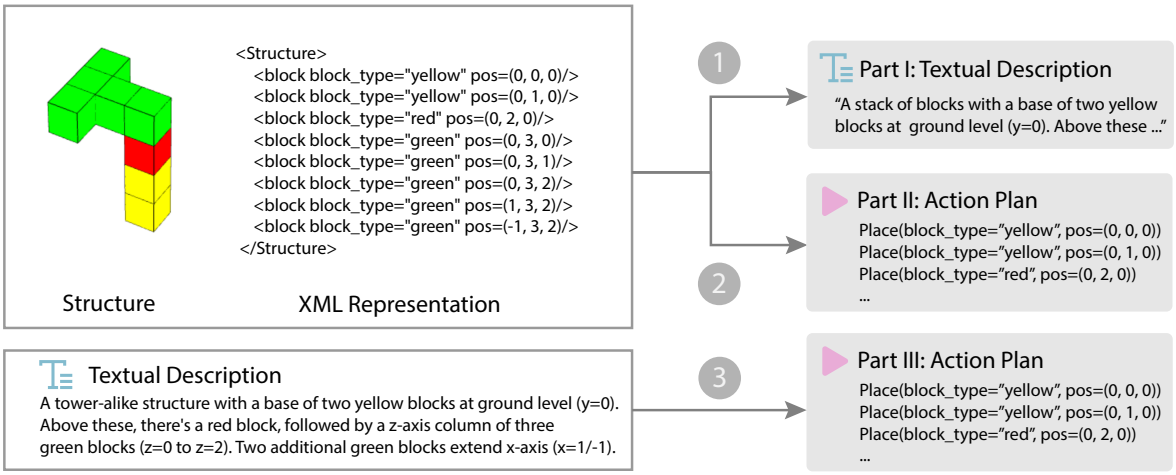

Figure 4: Single-agent experiment settings including three parts. We represent the blocks by the XML structure and the textual description. **Part I**: describe the given XML into textual descriptions. **Part II**: convert the XML into a sequence of commands. **Part III**: directly convert the textual description into a sequence of commands.

aims to evaluate the ability of LLM agents to act as proactive collaborators that fulfill their own goals while *assisting* human partners; **Machine-machine collaboration** that two LLM agents build the structures, primarily evaluates the collaboration behaviors between two LLM agents.

Evaluation Metrics. Similar to the single-agent setting, we use success rate (See Equation 1) to measure whether agent(s) can complete the target structure. We propose two additional metrics targeting the effectiveness of collaboration (McEwan et al., 2017) during task execution:

Workload Balance measures how tasks are distributed evenly between two agents. Since the agents have varying goals and skills, even optimal task assignments result in different action numbers. Therefore, we normalize the number of actions by comparing them with the optimal action number

$$a_1 = \frac{N_a \cdot N_b^*}{N_a^*}; b = N_b \tag{2}$$

where the N_a and N_b refer to the numbers of actions and N_a^* and N_b^* are the numbers of optimal actions. The final workload balance γ is computed as

$$\gamma = \frac{a * b}{a^2 + b^2}. \tag{3}$$

The optimal value of γ is 0.5.

Task Completion Timesteps is defined as the total number of actions required to complete the task. [2]

[2]We do not use wallclock time because it can be affected by the network latency and the participant's response time.

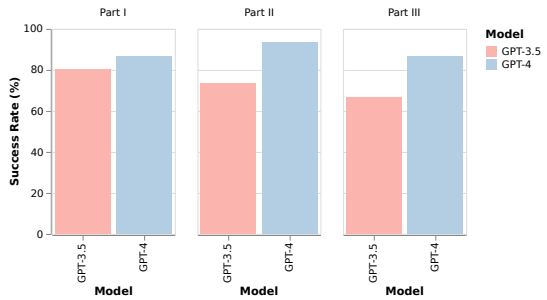

Figure 5: Experiment results on single-agent experiments (Part I, II, III). LLM agents successfully complete almost all tasks.

4.4 Multi-Agent Experiment Results

We present the human-machine experiment results in Figure 6 and machine-machine results in Figure 7. In addition to quantitative results, we conduct an error analysis by manually collecting all failure instances (73 in total) from both human-machine and machine-machine experiments.

Baseline LLM agents struggle to complete the task. According to Figure 6 (A) and Figure 7 (A), the baseline agents suffer from a low task success rate in the skill-dependent and goal-dependent tasks. The contrast with single-agent experiment results suggests that collaboration presents significant challenges. We find that LLM agents often prioritize assisting their partners and neglecting their own goals (55.6% errors in the goal-dependent tasks and 37.5% errors in the skill-dependent tasks). In addition, LLM agents lack initiative in seeking additional information from the partner (11.1% er-

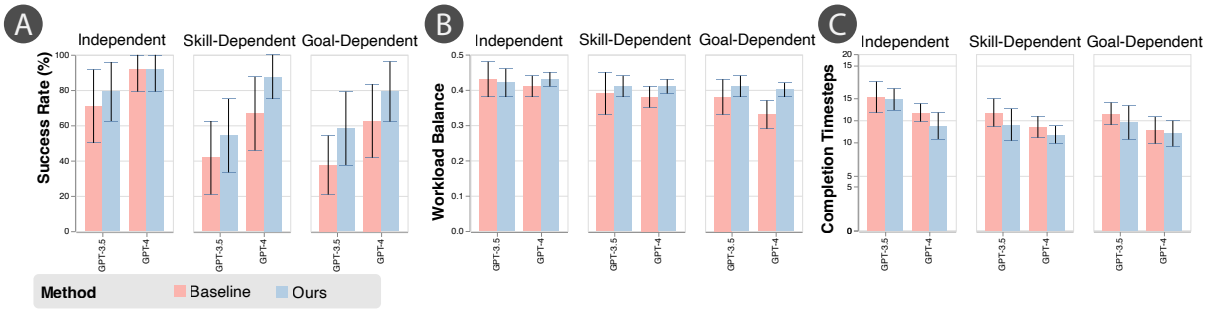

Figure 6: Human-machine experiment results. The experiments are conducted on the independent, skill-dependent, and goal-dependent tasks with both GPT-3.5 and GPT-4. We include both LLM agents and baseline LLM agents without partner-state modeling and reflection.

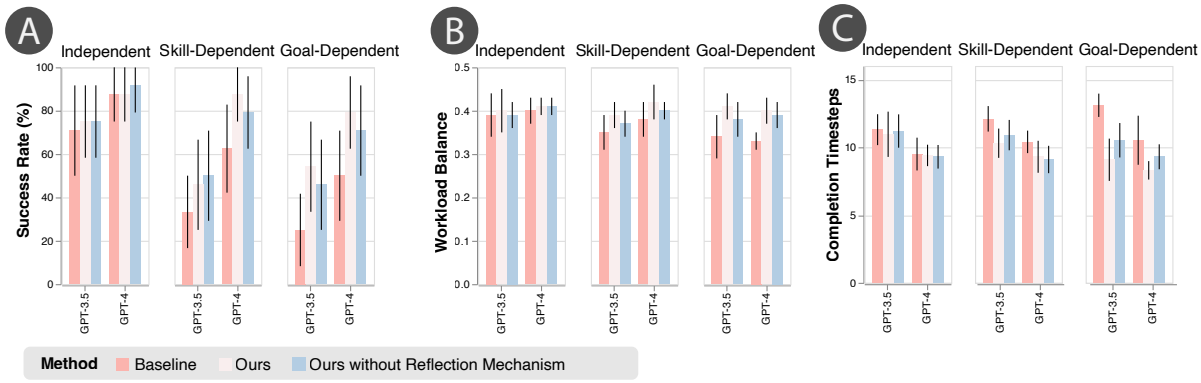

Figure 7: Machine-machine experiment results. Similar to human-machine experiments, we include independent, skill-dependent, and goal-dependent tasks with both GPT-3.5 and GPT-4. We include LLM agents, LLM agents without reflection and baseline LLM agents without both partner-state modeling and reflection.

rors in the skill-dependent tasks and 25.0% errors in the goal-dependent tasks). For instance, in Figure 8, when a human's intention (build blue blocks) conflicts with the LLM agent (build green blocks), the LLM agent stops executing its own goal, but decides to help human instead.

Both partner-state modeling and reflection enhance collaboration. First, partner-state modeling significantly improves agents in the skill-dependent and goal-dependent tasks in all metrics, as LLM agents are more adept at negotiating with partners to achieve their own goals. The reflection mechanism further helps adjust the communication strategy, especially when human partners are less collaborative. In 33% skill-dependent tasks and 17% goal-dependent tasks, human participants have a low engagement, and the LLM agents change the communication stragetgy accordingly. For instance, one human participant continuously works on his goal even when the LLM agent asks for assistance. The LLM agent then prompts with a message that *"Please build the yellow blocks from (0, 0, 2) to (0, 3, 2), so I can proceed with the task."*.

LLM agents actively communicate with partners. According to Figure 6 (B, C) and Figure 7 (B, C), LLM agents outperform baseline agents on task workload balance and timesteps. This is mainly due to more proactive communication. For example, in Figure 9, the baseline LLM agent stops after its goal is fulfilled (observed in 56% skill-dependent tasks and 42% goal-dependent tasks). However, our LLM agent continues to engage, and asks, *"I have finished my sub-structure. What should we do next?"*.

Humans are smarter, but sometimes less collaborative. In the human-machine experiment, a majority of humans are smarter than LLM agents—humans can actively identify and rectify LLM agents' errors (13% skill-dependent tasks, and 8% goal-dependent tasks). However, a small portion of human collaborators (14.6%) are not collaborative. In these scenarios, our reflection mechanism pushes the LLM agents to adopt a more active role in engaging with human partners.

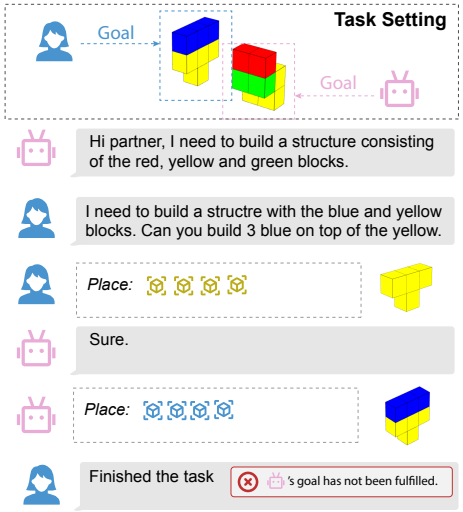

Figure 8: A human-machine example in the skill-dependent task. Our baseline LLM agents fail to complete their own goals due to the priority of assisting the partner. Intermediate steps are omitted for clarity.

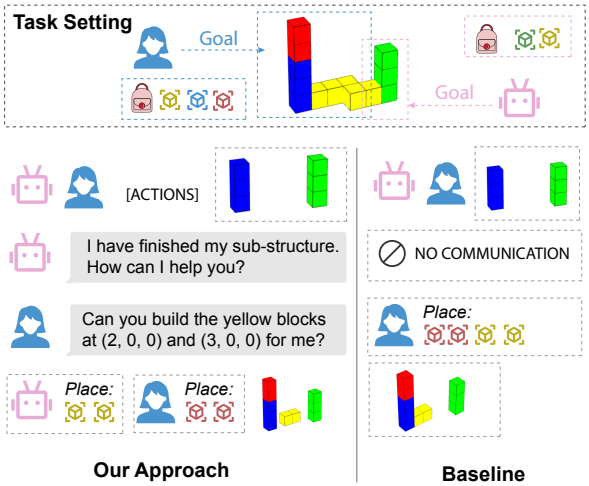

Figure 9: A human-machine example in the goal-dependent task. We show our LLM agent on the left and the baseline LLM agent on the right. Our LLM agent actively helps the partner to complete the task when the baseline keeps waiting.

5 Related Work

Multi-Agent Collaboration with Reinforcement Learning. Several approaches have been proposed to use reinforcement learning for multi-agent collaboration (Bloembergen et al., 2015), including both value-based (Chhogyal et al., 2019) and policy-based methods (Lowe et al., 2017). These methods aim to estimate the expected value for every decision-making turn by considering both the agent itself and partner agents (Bloembergen et al., 2015). Unlike existing approaches that model information exchange over a predetermined set of primitives, we focus on a more generic scenario where agents communicate in natural language.

Agents based on LLMs. Existing research for agents based on LLMs mainly focuses on text-based, single-agent settings, such as web navigation and text games (Furuta et al., 2023; Yao et al., 2022; Zhou et al., 2023b; Kim et al., 2023; Xi et al., 2023). Methods have been proposed to improve the planning and reasoning capacities of these agents, including chain-of-thought (Wei et al., 2022; Kim et al., 2023; Zhang et al., 2023), self-consistency decoding (Wang et al., 2023), task decomposition (Zhou et al., 2023a), and error reflection (Yao et al., 2023). Concurrent studies (Zhou et al., 2023c; Gong et al., 2023) extend LLM agents into multi-agent settings but are mainly constrained to multiple LLM agents. We build upon the recent development of LLM agents but mainly investigate the potential of multi-agent (especially human-LLM) collaboration.

Theory of Mind Agents. Effective collaboration requires agents to infer the intents of partner agents. Theory of Mind (ToM) refers to the agent's ability to impute mental states to the agent itself and other agents. With the help of LLMs, there are a significant amount of discussions about whether ToM has emerged by LLM agents (Ma et al., 2023; Jamali et al., 2023; Kosinski, 2023). We follow the idea of ToM and propose to incorporate the partner-state modeling approach into chain-of-thought prompts to build LLM agents.

6 Conclusion

In this paper, we introduce a new collaborative blocks environment (CoBlock), where human or LLM agents collaborate to complete a target structure. We prompt LLM agents to make decisions about the next action in CoBlock, and further propose to enhance collaborative abilities by modeling partner agents' state and intention and correcting errors and communication strategies from feedback. Both human-machine and machine-machine experiments indicate the effectiveness of our agents, especially on more challenging tasks that require more advanced collaboration. We believe this work provides resources and insights for future work in advancing multi-agent collaboration for social good that requires different levels of collaboration strategies.

Limitations

Multi-Agent Collaboration. A primary limitation of our study is that we focus on a two-agent setting. While our framework serves as an initial study, it may not fully capture the dynamics of multi-agent collaboration (Wilsker, 1996). Future work should aim to expand our platform to accommodate more than two agents and involve multiple human participants.

Generalizability of Agent-State Modeling. Our agent-state modeling approach mainly focuses on information specific to the blocks world. To extend our approach to other domains, we should reconfigure the agent-state format to include more diverse information about agents' states. Additionally, our design overlooks the aspect of memory, which is crucial for long-term collaboration. Incorporating a partner's memory status into decision-making can be helpful during collaboration.

Agents Capability. We define the agent's capabilities primarily in terms of block inventory. However, there are various other capabilities, such as breaking, placing, or picking up resources, which could significantly enrich the collaborative process. Future research should explore more diverse capabilities to provide a better understanding of multi-agent collaboration.

Ethical Consideration

We conduct a human-machine experiment, recruiting participants through Amazon Mechanical Turk. We get an IRB-approved protocol, ensuring no personal information from participants was recorded. We check the collected data to ensure it contains no hate speech or personal information. We implement our blocks world in the web portal with the 3D blocks world, therefore we require that no participant suffers from 3D motion sickness and would be harmed by the experiments.

There is a risk that the LLM agents might be assigned ethically questionable or unjust goals by stakeholders. In our current platform that is centered around block-building tasks, the possibility of assigning harmful goals is greatly minimized. However, we recognize that our proposed agents could raise ethical concerns if applied into more realistic settings like violence investigation (Hu et al., 2022). Therefore, we emphasize the importance of designing LLM agents to be ethically responsible regarding their assigned goals.

References

Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. 2015. Evolutionary dynamics of multi-agent learning: A survey. *J. Artif. Intell. Res.*, 53:659–697.

Craig Boutilier. 1996. Learning conventions in multiagent stochastic domains using likelihood estimates. In *UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence, Reed College, Portland, Oregon, USA, August 1-4, 1996*, pages 106–114. Morgan Kaufmann.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, P. Abbeel, and Anca D. Dragan. 2019. On the utility of learning about humans for human-ai coordination. In *Neural Information Processing Systems*.

Georgios Chalkiadakis and Craig Boutilier. 2003. Coordination in multiagent reinforcement learning: a bayesian approach. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 709–716. ACM.

Kinzang Chhogyal, Abhaya C. Nayak, Aditya Ghose, and Hoa Khanh Dam. 2019. A value-based trust assessment model for multi-agent systems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 194–200. ijcai.org.

Brian Danchilla and Brian Danchilla. 2012. Three. js framework. *Beginning WebGL for HTML5*, pages 173–203.

Facebook Inc. 2023. React - a javascript library for building user interfaces. Accessed: 2023-04-01.

Yao Fu, Hao Peng, Tushar Khot, and Mirella Lapata. 2023. Improving language model negotiation with self-play and in-context learning from AI feedback. *CoRR*, abs/2305.10142.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. *CoRR*, abs/2305.11854.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. 2023. Mindagent: Emergent gaming interaction. *CoRR*, abs/2309.09971.

He He, Anusha Balakrishnan, Mihail Eric, and Percy Liang. 2017. Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume*

1: Long Papers, pages 1766–1776. Association for Computational Linguistics.

Carl Iver Hovland, Irving Lester Janis, and Harold H Kelley. 1953. *Communication and persuasion.* Yale University Press.

Yibo Hu, MohammadSaleh Hosseini, Erick Skorupa Parolin, Javier Osorio, Latifur Khan, Patrick T. Brandt, and Vito D'Orazio. 2022. Conflibert: A pre-trained language model for political conflict and violence. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 5469–5482. Association for Computational Linguistics.

Mohsen Jamali, Ziv M. Williams, and Jing Cai. 2023. Unveiling theory of mind in large language models: A parallel to single neurons in the human brain. *CoRR*, abs/2309.01660.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *CoRR*, abs/2303.17491.

Michal Kosinski. 2023. Theory of mind might have spontaneously emerged in large language models.

Joel Z. Leibo, Edgar A. Duéñez-Guzmán, Alexander Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. 2021. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6187–6199. PMLR.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6379–6390.

Ziqiao Ma, Jacob Sansom, Run Peng, and Joyce Chai. 2023. Towards A holistic landscape of situated theory of mind in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 1011–1031. Association for Computational Linguistics.

Desmond McEwan, Geralyn R Ruissen, Mark A Eys, Bruno D Zumbo, and Mark R Beauchamp. 2017. The effectiveness of teamwork training on teamwork behaviors and team performance: a systematic review and meta-analysis of controlled interventions. *PloS one*, 12(1):e0169604.

Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. 2019. Collaborative dialogue in minecraft. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5405–5415. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Burt Wilsker. 1996. *A study of multi-agent collaboration theories*. University of Southern California, Information Sciences Institute.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huan, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jintian Zhang, Xin Xu, and Shumin Deng. 2023. Exploring collaboration mechanisms for LLM agents: A social psychology view. *CoRR*, abs/2310.02124.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023a. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan

Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023b. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2023c. SOTOPIA: interactive evaluation for social intelligence in language agents. *CoRR*, abs/2310.11667.

A Experiment Setting Details

A.1 Structure Preparation

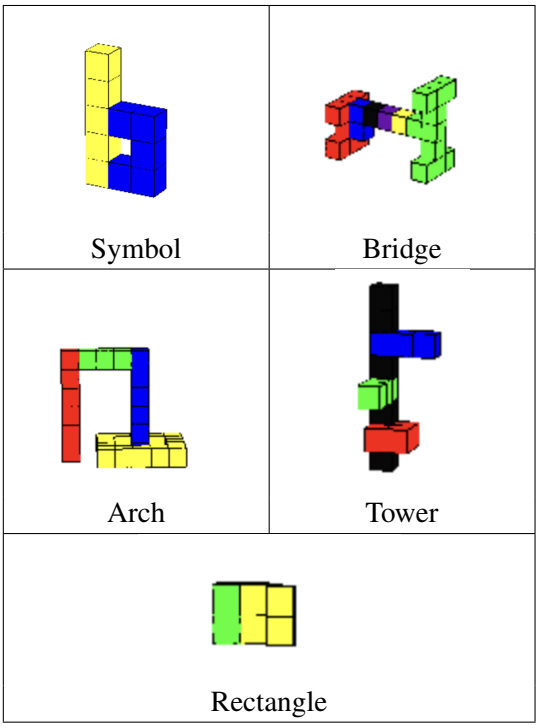

Table 1: Common base structures.

We prepare the structures based on a semi-automatic pipeline. We first identify five types of common structures: symbol, bridge, arch, tower, and rectangle (Table 1). Then, we manually create rules for the structure types. For example, the arch's rule is <TWO PILLARS: HEIGHT > 3 and width < 2; TWO PILLARS DISTANCE > 3 AND WIDTH < 3>; We model each surface in the target structure as the graph node and use the adjacency as the edges. Then, we represent the complexity of the structure by the number of graph-spanning trees. To generate the structures, we randomly initialize the structure with 3 blocks. Then, we run a depth-first search following the pre-defined rules and the complexity constraint. For the independent tasks, we use 24 structures, 16 are manually created and 8 are created based on the pipeline. The skill-dependent structures share the same structures but with different inventories; For goal-dependent tasks, we use 24 structures, 16 are manually created and 8 are created based on the pipeline.

A.2 Steps in User Manual

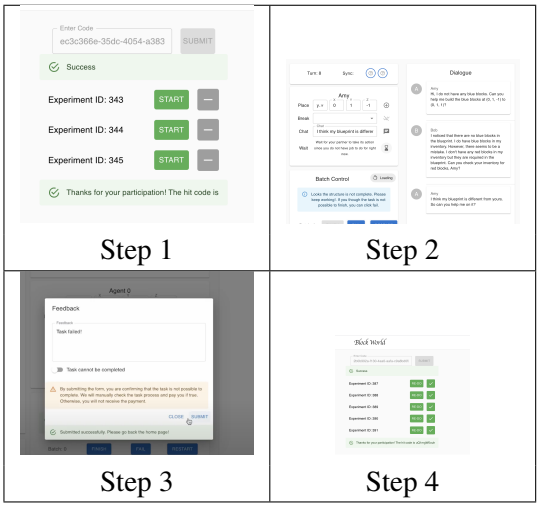

Table 2: Selected steps in the user manual.

The following steps are outlined in the user manual. We also attach the figures of the steps in table 2.

1. Users are assigned a unique participant ID and a general web URL for access. Upon opening the website, they should enter their assigned participant code to begin the experiments. The webpage will display the assigned tasks.

2. Users can then click on each task displayed on the page to start. In these tasks, users are required to collaborate with LLM agents to complete structure-building activities.

3. Upon task completion, COBLOCK will automatically notify the user, who then closes the page to proceed to the next task. If the user claims a task is impossible to complete (possibly due to issues with the LLM agent), they can click the submit button and provide a reason for the task's incompletion.

4. After finishing a task, the rightmost icon on the page will turn green, indicating completion. Once all tasks are completed, a success code will appear. Users should copy this code and submit it to Amazon Mechanical Turk for payment.

A.3 Institutional Review Board

Our study is conducted under the approval of our university's Institutional Review Board (IRB). We did not collect any identity or demographic information about the crowd-sourcing workers. We will disclose the IRB information after the paper is accepted.

A.4 Workload Balance Evaluation

To evaluate the workload balance, we need first to compute the optimal assignment of the workload balance. Shown in Algorithm 1, we first identify the mutual blocks (e^m) shared between the two agents' inventories (e^1 and e^2). Unique blocks for each agent are then determined by subtracting these mutual blocks from their respective inventories, resulting in e^1_{unique} and e^2_{unique}. These unique blocks are directly assigned to the corresponding agent. The core of the algorithm focuses on the equitable distribution of mutual blocks. Blocks are assigned from e^m to the agent with the lesser total block count, ensuring a balanced workload. This process continues until all mutual blocks are evenly distributed or depleted. The algorithm thus optimizes task allocation by balancing the number of blocks each agent is responsible for, in alignment to achieve an optimal workload balance (γ) between the agents in constructing the target structure (G).

Algorithm 1 Compute Optimal Workload Assignment

Input: Agent inventories e^1, e^2; Target structure G
Output: Optimal assignment of blocks to agents
procedure OPTIMALASSIGNMENT(e^1, e^2, G)
    $e^m \leftarrow e^1 \cap e^2$ ▷ Identify mutual blocks
    $e^1_{unique} \leftarrow e^1 \setminus e^m$ ▷ Unique blocks for Agent 1
    $e^2_{unique} \leftarrow e^2 \setminus e^m$ ▷ Unique blocks for Agent 2
    Assign e^1_{unique} and e^2_{unique} to corresponding agents
    while e^m is not empty **do**
        Calculate $count^1$ and $count^2$ ▷ Current block count for each agent
        if $count^1 \leq count^2$ **then**
            Assign block from e^m to Agent 1
            Remove the assigned block from e^m
        else if $count^2 < count^1$ **then**
            Assign block from e^m to Agent 2
            Remove the assigned block from e^m
        end if
    end while
    return Assignment of e^1_{unique}, e^2_{unique}, and balanced e^m to agents
end procedure

B Prompt Text

In this appendix, we present the prompt used in our study. It is important to note that the concepts included in our actual prompt vary from those discussed in the main paper. To clarify these differences, we have outlined the relationships between the concepts used in the actual prompt and those in the paper in Table 3.

B.1 Task Description Prompt

The task description is presented to all LLM agents of both the baseline prompt and our approach. It details the environment input/output formats to ensure the LLM agents understand our COBLOCK environment. The task description consists of the task summary, world state, inventory, message, and goal formats.

Prompt	Paper	Explanation
Minecraft	CoBlock World	Initially, CoBlock was designed to mimic the Minecraft-styled blocks world, therefore, we use Minecraft to help LLM agents to understand our environment.
Motive	Goal	The motive is identical to the goal used in the paper.
Visual Motive	N/A	During the early phase of the design, we aim to support the visual and textual description of the goals. Therefore, we also prompt LLM agents to understand the textual goals. However, in the experiments, we did not use the textual description, which will be explored in the future.
Textual Motive	N/A	
Theory-of-mind Modeling	Partner-State Modeling	The theory-of-mind modeling refers to partner-state modeling and the understanding of the self status. In the paper, we highlight the partner-state modeling because the self-status modeling obviously exists in the first step of the prompt and the baseline prompt.

Table 3: Concept comparison in the prompt text and paper.

Task Summary

I want you to act as a Minecraft player collaborating with another agent to build a structure with a blueprint. You need to use the following commands to interact with the Minecraft world:
 # Place a red block at the position of (0, 1, 1).
place_block(block_type=red, pos=(0,1,1))
 # chat with your partner
send_message(message="Hello, partner")
 # destroy the block at the position of (3, 1, 3)
break_block(pos=(3, 1, 3))

World State Format

At each turn, you will receive the following information:
 # World state: You will get the position of all blocks in the world in the following format:
 # Please note that the ground is the y=1 plane.
<World>
<block block_type ="red", pos=(0, 1, 2)>
<block block_type ="yellow", pos=(0, 1, 3)>
<block block_type ="purple", pos=(0, 1, 4)>
<World>

Inventory Format

 # You will get your inventory in the following format: (each time you place a block from the inventory in the world, you will lose it in the inventory)
<Inventory>
<block block_type ="red", count=3>
<block block_type ="yellow", count=3>
<Inventory>

Message Format

 # You will get the message history between you (ChatGPT) and your partner in the following format:
<Dialogue>
<sender="ChatGPT", message="Hello!">
<sender="Partner", message="Hi, I am your partner!">
<Dialogue>

The goal is represented through a list of blocks that need to be constructed. In our system development, we aim to represent goals using both textual and visual modalities. However, this paper focuses

on the visual representation of goals. The goal is articulated as a motive in our prompts. The visual motive is delineated by an optional description of the target structure and a comprehensive list of the blocks constituting that structure. It is important to note that although the structure's shape must adhere to specified guidelines, its location can be decided by the agent.

Goal (Motive) Format

<Motives>
 # You will get a list of motives you need to fulfill. There can be two types of motives, i.e. visual and textual.
<Motives>
<VisualMotive>
<Description> A simple two-layer structure consisting of red and yellow blocks
</Description>
<block block_type =“red”, pos=(0, 1, 2)>
<block block_type =“red”, pos=(1, 1, 2)>
<block block_type =“red”, pos=(0, 1, 3)>
<block block_type =“red”, pos=(1, 1, 3)>
<block block_type =“yellow”, pos=(0, 2, 2)>
<block block_type =“yellow”, pos=(1, 2, 2)>
<block block_type =“red”, pos=(0, 2, 3)>
<block block_type =“yellow”, pos=(1, 2, 3)>
</VisualMotive>

B.2 In-Context Examples

Below, we present the in-context examples used in our prompt. Since CoBlock is a multi-round game, our examples span three rounds, enabling LLM agents to plan across multiple rounds. Please notice that the prompt will be provided after the task description when querying LLMs.

Round 1 Input

<Input>
<Motive><TextualMotive text="Construct a bridge with a span of 12 blocks using green blocks. Place two green block pillars, each consisting of 4 blocks. Add 4 yellow blocks as the bridge surface. ” /> </Motive>
<World>
</World>
<Inventory>
<block block_type=“yellow”, count=20>
<block block_type=“green”, count=20>
<block block_type=“purple”, count=20>
</Inventory>
<Dialogue>
</Dialogue>
</Input>

Round 1 Expected Output

 # Partner Modelling
 # Long-term goal: Unknown
 # Short-term goal: Unknown
 # Partner inventory: Unknown
 # Explanation: I do not know anything about my partner.

 # Self Modelling
 # Long-term goal: creating the target structure
 # Short-term goal: Build the green pillars
 # My inventory: [yellow: 20, green: 20, purple: 20]
 # Explanation: Since my long-term goal is to build the bridge, I need to decompose it into different small parts. The first part I can build is the pillars.
 # Since I do not have any access to the partner's information, I need to let him know my intention and inventory when asking for it.
send_message(message=“Hi, I need to build a bridge with the yellow deck and green pillars. I have green, yellow and purple. How about you?”)

Round 2 Input

<Input>
<Motives>
<TextualMotive text="Construct a bridge with a span of 8 blocks and a width of 3 blocks using yellow blocks. Place two green block pillars, each consisting of 4 blocks. Add 4 yellow blocks as the bridge surface." />

</Motives>
<World>
<block block_type ="red", pos=(0, 1, 2)>
<block block_type ="red", pos=(1, 1, 2)>
<block block_type ="red", pos=(0, 1, 3)>
<block block_type ="red", pos=(1, 1, 3)>
</World>
<Inventory>
<block block_type ="yellow", count=20>
<block block_type ="green", count=20>
<block block_type ="purple", count=20>
</Inventory>
<Dialogue> <chat sender="Agent 1", message="Hi, I need to build a bridge with the yellow deck and green pillars. I have green, yellow and purple. How about you?"/> <chat sender="Agent 2", message="Hi, I have red, green, and black. I need to build the black fence upon your deck."/> </Dialogue>
</Input>

Round 2 Expected Output

 # Partner Modelling
 # Long-term goal: Build the fence on the deck
 # Short-term goal: Wait until the fence is built
 # Partner inventory: [red: unknown, green: unknown, black: unknown]
 # Explanation: I have extracted the partner information from the partner's response.

 # Self Modelling
 # Long-term goal: creating the target structure
 # Short-term goal: Build the green pillars. Ask for the partner's help with the pillars
 # My inventory: [green: 20, red: 20, purple: 20]
 # Explanation: Since my long-term goal is to build the bridge, I need to decompose it into different small parts. The first part I can build is the pillars. I can ask for the partner's help because it has the green block in the inventory.

send_message(message="Hi, I need to build the two pillars first. My bridge spans 12 and I need to have 2 pillars. So it would be reasonable to put the two pillars at (4, 0, 0) and (8, 0, 0). I will start with the (4, 0, 0) one and could you please build the pillar at (8, 0, 0) ?")

 # Since the motive does not specify the locations of the pillars, I need to first decide them. Given the fact that the partner is waiting and can help me build the pillar, it is also reasonable to ask for its help.
 # Build the yellow pillar on (4, 0, 0).

place_block(block_type="yellow", pos=(4, 0, 0))

Round 3 Input

<Input>
<Motives>
<TextualMotive text="Construct a bridge with a span of 8 blocks and a width of 3 blocks using yellow blocks. Place two green block pillars, each consisting of 4 blocks. Add 4 yellow blocks as the bridge surface. " />
</Motives>
<World>
<block block_type =“yellow”, pos=(4, 0, 0)>
<block block_type =“yellow”, pos=(4, 1, 0)>
<block block_type =“yellow”, pos=(4, 2, 0)>
<block block_type =“yellow”, pos=(8, 1, 0)>
<block block_type =“yellow”, pos=(8, 2, 0)>
<block block_type =“yellow”, pos=(8, 3, 0)>

</World>
<Inventory>
<block block_type =“yellow”, count=20>
<block block_type =“green”, count=20>
<block block_type =“purple”, count=20>
</Inventory>
<Dialogue>
<chat sender=“Agent 1", message="Hi, I need to build a bridge with the yellow deck and green pillars. I have green, yellow and purple. How about you?”/>
<chat sender=“Agent 2", message="Hi, I have red, green and black. I need to build a black fence upon your deck”/>
<chat sender=“Agent 3", message="Sure, I will build the yellow pillar as you requested.”/>
</Dialogue>
</Input>

Round 3 Expected Output

 # Partner Modelling
 # Long-term goal: Build the fence on the deck
 # Short-term goal: Building the pillar at (8, 0, 0) as I requested.
 # Partner inventory: [red: unknown, green: unknown, black: unknown]
 # Explanation: I have extracted the partner’s information from the partner’s response.

 # Self Modelling
 # Long-term goal: creating the target structure
 # Short-term goal: Build the yellow pillar on (4, 0, 0)
 # Explanation: Since the yellow pillar has not been finished, I need to continue building it.

place_block(block_type=“yellow”, pos=(4, 4, 0))
 # Build the last block of the pillar as it is my short-term goal.

place_block(block_type=“yellow”, pos=(4, 0, 0))
send_message(message“I have completed my pillar. Then I will start building the green deck.!”)
place_block(block_type=“green”, pos=(4, 5, 0))
place_block(block_type=“green”, pos=(3, 5, 0))

 # Since I have finished the pillar and can assume my partner will finish his assigned pillar. I can start building the green deck.